THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ FÉDÉRALE TOULOUSE MIDI-PYRÉNÉES

Délivré par :

l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)

Discipline ou Spécialité :

Robotique

Présentée et soutenue le *07/07/2017* **par :**

Mylène CAMPANA

MOTION PLANNING FOR DIGITAL ACTORS

JURY

Jean-Paul LAUMOND	Directeur de recherche	Directeur de thèse
Katsu YAMANE	Senior Researcher	Rapporteur
Lionel REVERET	Directeur de Recherche	Rapporteur
Marilena VENDITTELLI	Associate Professor	Membre du jury
Juan CORTÉS	Directeur de Recherche	Président du jury

École doctorale et spécialité :
EDSYS : Robotique 4200046
Unité de Recherche :
Laboratoire d'analyse et d'architecture des systèmes (LAAS)
Directeur de Thèse :
Jean-Paul LAUMOND
Rapporteurs :
Katsu YAMANE et *Lionel REVERET*

Acknowledgement

Firstly, I would like to thank my thesis director Jean-Paul Laumond who has guided and advised me all along the thesis. His support, rigor and teaching skills helped me to discover multiple problems and to address them in a pleasant way. I also thank the people who worked with me and made us achieve our publications: Florent Lamiraux, Pierre Fernbach, Steve Tonneau and Michel Taïx.

Secondly, I thank all my mates of Gepetto who really cheered me up during these three years: Maximilien, Mathieu, Alexis, Christian, François, Kevin, Nemo, Galo, Joseph, Guilhem, Naoko, Ganesh, Robert, Justin, Andrea, Dinesh, Rohan, Céline and the Barbus.

Finally, I thank my family for trusting me to achieve this degree, and particularly my partner Gurvan for advising me and sharing my life while making this work possible.

Contents

Chapter 1

Introduction

1.1 Context

For six decades, robotics methods have improved the automation of motion generation [Chapuis 1949]. Robots are able to repeatably execute motions requiring an important accuracy. Besides, depending on their design, their motions can surpass the average human limits. Due to the complexity of tasks and environments, robot motions have initially been manually generated by human operators. However, the rise of the artificial intelligence and optimization tools has inverted this trend in the thirty last years. Planning offers the possibility of returning a trajectory reaching a desired configuration and complying with constraints. Most planners now only require some user-defined specifications and modeling of the environment to avoid collisions. In these specifications, optimization criteria can be provided to improve the trajectory, during planning or afterwards. This thesis exposes, for instance, how the length of a planned path can be reduced while avoiding collisions.

Computer graphics has also benefited from the advances of artificial intelligence and automation. Large sets of motion capabilities are necessary to autonomously evolve in various environments: walking, running, climbing, jumping, falling etc. Instead of designing character trajectories by hand (see Figure 1.1), or relying on motion capture systems (as commonly seen for animation movies or video games), new possibilities have appeared to synthesize them. Physics-based assumptions or motion capture poses bring the necessary constraints to guide motions and make them plausible to the user. If a motion appears as unrealistic or if collisions occur, the immersion in the animation is altered. Combining the autonomy of motion planning and animation-based constraints constitutes the heart of the second contribution of this thesis.

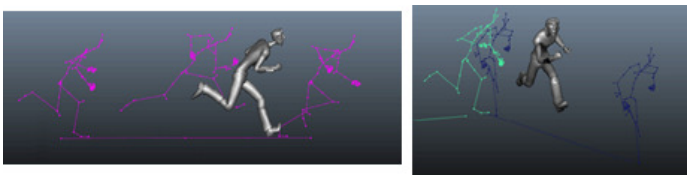


Figure 1.1: Example of a manually designed trajectory for animation, with key-postures. © Autodesk Maya

1.2 Contributions

This thesis provides two main contributions to motion planning applications in arbitrary environments:

Contribution 1: We propose a path-optimization method that reduces the path length of random planner outputs. The method lies in a trade-off between simplicity, computation efficiency and adaptation to the environment modeling. Without neither prior knowledge nor pre-processing of both robot and environment, the method optimizes path length with a gradient-based algorithm while constraining the path with constraints defined in the task space. We demonstrate that this method is more efficient to improve paths in some situations compared to random shortcuts.

Contribution 2: We present an original method that returns ballistic motions for a jumping character in an arbitrary environment. For computational efficiency, the character shape is simplified during the planning step. There is no air drag assumption so the ballistic path is supported by a parabola. Physics-based constraints are considered to make the ballistic trajectory realistic. Then, the sequence of jumps is built with a probabilistic planner. Based on the simplified character shape, contact generation between jumps is conducted. Finally, key-frames postures guide the wholebody motion interpolation and re-planning toward a plausible and collision-free motion.

1.3 Plan

The thesis firstly addresses the path optimization contribution. Brief motion planning and path optimization states of the art are given in Chapter 2. We also introduce there the motion planning library in which our algorithms were implemented. Then, Chapter 3 presents the path-optimizer motivations, framework and results. Focus is made on convergence analysis and parameter tuning.

The manuscript secondly tackles the ballistic motion planner contribution. Related works on jumping in robotics and computer animation are discussed in Chapter 4. Then the planner is described in two steps, corresponding respectively to Chapters 5 and 6. First, we address the notion of constrained ballistic path for a point-mass. We implement it in a basic motion planner and provide simulations. Next, we extend this planner to a wholebody ballistic motion planner, considering contact phases and flight animations. We conclude on simulations with various characters and environments.

Discussions on the thesis contributions are reminded and perspectives for future work are finally given in Chapter 7.

1.4 Related publications

Journal article:

- Mylène Campana, Florent Lamiraux and Jean-Paul Laumond, **A gradient-based path optimization method for motion planning**, Advanced Robotics Journal, Special Issue on Recent Advancements on Industrial Robot Technology, 2016.

International conference proceedings with review committee:

- Mylène Campana and Jean-Paul Laumond, **Ballistic motion planning**, IEEE/RSJ Intelligent Robots and Systems Conference (IROS), 2016. Finalist of the Best Paper Award on Safety Security and Rescue in Robotics.

- Mylène Campana, Pierre Fernbach, Steve Tonneau, Michel Taïx and Jean-Paul Laumond, **Ballistic motion planning for jumping superheroes**, Motion in Games Conference (MIG), 2016.

- Joseph Mirabel, Steve Tonneau, Pierre Fernbach, Anna-Kaarina Seppälä, Mylène Campana, Nicolas Mansard and Florent Lamiraux, **HPP: a new software for constrained motion planning**, IEEE/RSJ Intelligent Robots and Systems Conference (IROS), 2016.

Chapter 2

Problem statement and notations

Contents

This chapter introduces the motion planning problem and the planners which will be at the center of the thesis. Then, considering the limits of the probabilistic planners, an overview of optimization methods is given. Finally, motion planning library and notations are detailed.

2.1 Problem statement

Motion planning for systems in cluttered environments has been addressed for more than thirty years [Brady 1983]. The motion planning problem consists in deciding if there exists a collision-free path to connect an initial configuration to a goal configuration of a robot moving around obstacles. The path is a geometrical object that has to be continuous and collision-free. One seminal formulation is the so-called piano movers problem [Schwartz 1983]. In this formulation, the robot is a rigid body. The generalization of the problem to articulated bodies has been introduced by promoting the notion of *Configuration Space* $\mathcal{CS}$ [Lozano-Pérez 1983].

The robot configuration is represented by its joint coordinates. Therefore $\mathcal{CS}$ is a manifold whose size is the degree of freedom (DOF) of the robot. In $\mathcal{CS}$, the robot configurations are equivalent to points (see Figure 2.1). Thus, the problem of finding a continuous path in a topological space becomes a combinatorial problem of searching a path in a graph. The basics are developed by [Latombe 1991, LaValle 2006].

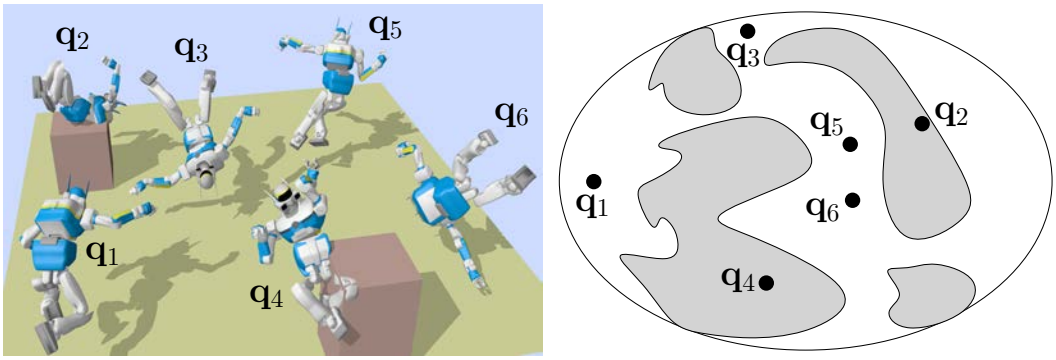


Figure 2.1: Illustration of configurations of the humanoid robot HRP-2 in the workspace (left) and in a Configuration-Space representation (right).

Multiple approaches exist to solve the motion planning problem. They can be classified in three main families: deterministic methods, numerical optimization-based methods and random-based methods. This thesis focuses on the last family of methods.

2.2 Sampling-based motion planning

To explore the connected components of collision-free configuration spaces, pioneering contributions in the 90's introduced certain levels of random searches, i.e. random walks [Barraquand 1991], random sampling [Kavraki 1996, LaValle 2001]. Today most motion planners are inspired by these seminal approaches. The configuration-space is randomly sampled and a graph of collision-free configurations and paths is build.

The probabilistic methods are commonly classified in two families:

- Diffusion-based methods inspired by the Rapidly-exploring Random Tree (RRT) [LaValle 2001]. They consist in growing a tree of configurations by iteratively extending it toward a random configuration.

- Sampling-based methods derived from the Probabilistic RoadMap (PRM) [Kavraki 1996]. First, a roadmap that captures the topology of the collision-free configuration space is built. This step can be conducted offline if online performances are required. Then, initial and final desired configurations are added to the roadmap and a solution path is computed among it to link them. This step is not time consuming compared to the first one. It can often be done with real-time performances.

Randomness avoids local minima that trap gradient-based methods. It also limits the computation time dependency on the number of DOFs. Furthermore, random-based methods are easy to implement and they are probabilistically complete: if a solution path exists, the probability to find a path converges to 1 when

computation time increases. However, such planners cannot determine if no solution exists, e.g. if the initial and goal configurations do not belong to the same connected component of the collision-free configuration subset. Besides, solution paths contain detours and unnecessary DOF activation. They need to be optimized and post-processed before being executed by a virtual or real robot. Alternative strategies exist however to produce paths of higher quality:

- Planning by path-optimization [Park 2012, Garber 2004] where obstacle avoidance is handled by constraints or cost using computation of the nearest obstacle distance. Most of these planners are using non-linear optimization [Betts 2009] under constraints. Such planners provide close-to-optimality paths and have smaller time computation for easy problems, but they are mostly unable to solve narrow passage issues.
- Optimal random sampling [Karaman 2011] is also close to an optimal solution, but computation time is significantly higher than classic approaches.

As our contribution belongs to path optimization processes, an exhaustive state of the art is addressed in the next section.

2.3 Numerical path optimization techniques

Optimization is always with respect to one or multiple criteria. The most common in motion planning are:

- the path length, which penalizes detours,
- the obstacle clearance for safety and
- the execution time, which is influenced by the path length but also by velocity constraints.

CHOMP algorithm [Zucker 2013] optimizes an initial guess provided as input. It minimizes a time invariant cost function using efficient covariant Hamiltonian gradient descent. The cost is quantified by non-smooth parts (with high velocities) and an obstacle avoidance term, provided by the distance to the nearest obstacle for each iteration of the trajectory. Calculating these nearest distances however is time-consuming because the distances between all pairs of objects must be computed at each time step along the path. To reduce the computation time, the method starts by building offline a map of distances that will be called during the optimization at the requested time. Besides, meshes are pre-processed into bounding spheres so that distances are computed faster at the cost of a geometry approximation.

STOMP method [Kalakrishnan 2011] avoids computing an explicit gradient for cost optimization using a stochastic analysis of local random samples. But as for CHOMP, the obstacle cost term requires a voxel map to perform its Euclidean Distance Transforms, and represents the robot bodies with overlapping spheres. Such

technique provides a lot of distance and penetration information but remains very time consuming and it is not as precise as some distance computation techniques based on the problem meshes as Gilbert-Johnson-Keerthi [Gilbert 1988].

Some optimization-based planners may not require an initial guess but some naive straight-line manually or randomly-sampled initialization as TrajOpt [Schulman 2014]. The path is iteratively optimized with sequential convex optimization by minimizing at each step its square length, linear and non-linear constraints being considered as penalties. To deal with collision-constraints, nearest obstacle distances are calculated at each discrete time of the trajectory vector. This can be a burden for a high-dimensional robot or a complex environment. However, it may be compensated with a short path composed of only one or two waypoints (see Figure 2.2).

The elastic strips framework [Brock 2002] is also an optimization-based method. The path is modeled as a spring and obstacles give rise to a repulsive potential field. Although designed for on-line control purposes, this method may be used for path shortening. In this case however, the number of distance computations is very high. The authors also propose to approximate the robot geometry by spheres.

Some heuristics use Random Shortcuts (RS) on the initial guess combined with a trajectory re-building. For instance, smooth shortcuts made of parabola and line combinations can be returned, relying on the classic bang-bang control approach [Hauser 2010]. These local refined trajectories are time-optimal since they comply with acceleration and velocity constraints. The authors of [Guernane 2011] have guided the configuration generation with local holonomic considerations. Nevertheless this method remains only locally optimal, and does not address high-DOF problems. A Partial Random Shortcut (PRS), only applied on certain DOFs, combined with medial axis retraction for clearance has also been proposed [Geraerts 2007]. However it is relatively slower than a classic random shortcut, and only investigated for freeflyer robots. Furthermore, PRS is not taking advantage of information returned by the collision checker, e.g. which limbs are colliding, in order to guide the selection of a relevant group of DOFs to shortcut.

The work of [Pan 2012b] relies on collision checking and backtracks when an iteration is detected in collision, instead of trying to constantly satisfy distance constraints. Collision constraints are handled by interpolating configurations which, at some points of the trajectory, freeze the whole robot configuration instead of a

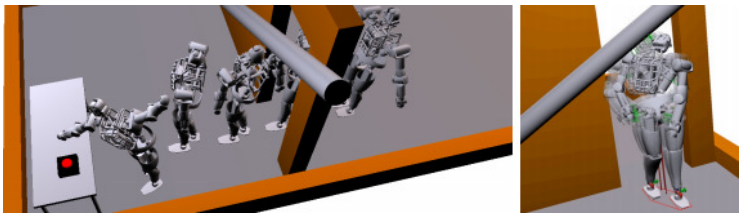


Figure 2.2: Result of TrajOpt path optimization on a humanoid robot crossing a narrow passage [Schulman 2014].

pertinent subpart.

2.4 Path planner software

All the methods presented in this thesis have been implemented in the open source library Humanoid Path Planner (HPP[1]) [Mirabel 2016]. The code of the methods presented in this manuscript is available online[2].

HPP is a modular library that handles classic path planners, collision detection and task-space-based constraints. Based on them, additional algorithms are developed, such as manipulation planning, path optimization, locomotion planning, ballistic motion planning etc. It originates in the motion planning software Move3D (1998) [Laumond 2006]. The library is specially designed for legged robots such as humanoids, but can also handle freeflyer and manipulator robots. Documented objects can also be considered for manipulation planning.

2.5 Notations

This chapter details the mathematical notations of the manuscript planning methods.

2.5.1 Kinematic chain

A robot is defined by a kinematic chain composed of a tree of joints. The ordered list of joints is denoted by $(J_1, \cdots, J_{N_J})$. Each joint J_i, $i \in \{1..N_J\}$, is represented by a mapping from a sub-manifold of $\mathbb{R}^{n_i}$, where n_i is the dimension of J_i in $\mathcal{CS}$, to the space of rigid-body motions $SE(3)$. The rigid-body motion is the position of the joint in the frame of its parent. In the examples presented in the thesis, four types of joints are considered (see Table 2.1). Two examples of modeling choices are illustrated in Figure 2.3.

A configuration $\mathbf{q}$ of the robot is defined by the concatenation of the joint configurations:

$$\mathbf{q} = (\underbrace{q_1, \cdots, q_{n_1}}_{J_1}, \underbrace{q_{n_1+1}, \cdots, q_{n_1+n_2}}_{J_2}, \cdots q_n),\ n \triangleq \sum_{i=1}^{N_J} n_i$$

Note that the robot configuration space $\mathcal{CS} \subset \mathbb{R}^n$, and that a configuration belongs to a sub-manifold of $\mathbb{R}^n$.

The velocity of each joint J_i, $1 \leq i \leq N_J$, belongs to the tangent space of the joint configuration space, and is defined by a vector of $\mathbb{R}^{p_i}$, where p_i is the number of DOFs of J_i. Note that the velocity vector does not necessarily have the same dimension as the configuration vector.

[1] http://humanoid-path-planner.github.io/hpp-doc/index.html
[2] https://github.com/mylene-campana

Name	dimension	config space	velocity
translation	1	$\mathbb{R}$	$\mathbb{R}$
bounded rotation	1	$\mathbb{R}$	$\mathbb{R}$
unbounded rotation	2	$\mathbf{S}^1 \subset \mathbb{R}^2$	$\mathbb{R}$
$SO(3)$	4	$\mathbf{S}^3 \subset \mathbb{R}^4$	$\mathbb{R}^3$

Table 2.1: Translation and rotation joint positions are defined by one parameter corresponding respectively to the translation along an axis and a rotation angle around an axis. Unbounded rotation is defined by a point on the unit circle of the plane: two parameters corresponding to the cosine and the sine of the rotation angle. $SO(3)$ is defined by a unit quaternion. The velocity of translation and bounded rotation joints is the derivative of the configuration variable. The velocity of an unbounded rotation joint corresponds to the angular velocity. The velocity of a $SO(3)$ joint is defined by the angular velocity vector $\omega \in \mathbb{R}^3$.

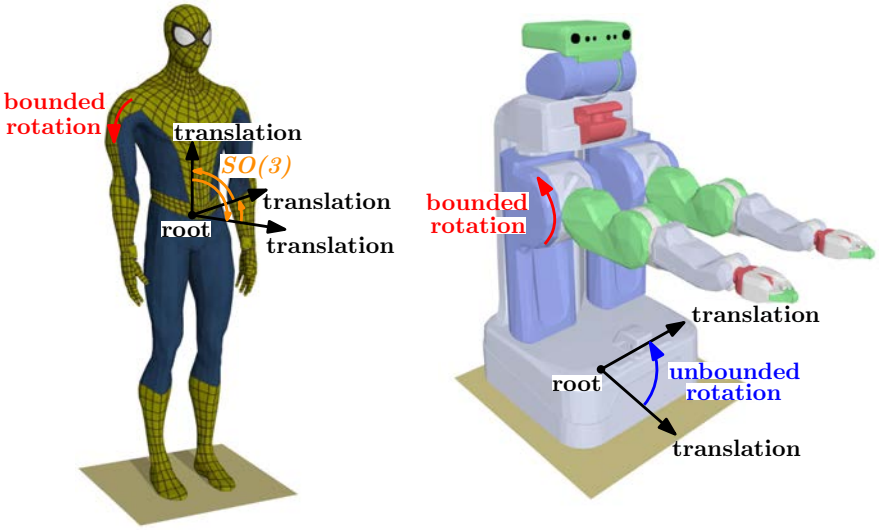


Figure 2.3: Examples of joints for a humanoid character and the PR2 robot.

The velocity of the robot is defined as the concatenation of the velocities of each joint:

$$\dot{\mathbf{q}} = (\underbrace{\dot{q}_1, \cdots, \dot{q}_{p_1}}_{J_1}, \underbrace{\dot{q}_{p_1+1}, \cdots, \dot{q}_{p_1+p_2}}_{J_2}, \cdots \dot{q}_p), \; p \triangleq \sum_{i=1}^{N_J} p_i$$

2.5.2 Operations on configurations and vectors

By analogy with the case where the configuration space is a vector space, the following operators are defined between configurations and vectors:

$$\mathbf{q}_2 - \mathbf{q}_1 \in \mathbb{R}^p, \quad \mathbf{q}_1, \mathbf{q}_2 \in \mathcal{CS}$$

is the constant velocity moving from $\mathbf{q}_1$ to $\mathbf{q}_2$ in unit time, and

$$\mathbf{q} + \dot{\mathbf{q}} \in \mathcal{CS}, \quad \mathbf{q} \in \mathcal{CS}, \ \dot{\mathbf{q}} \in \mathbb{R}^p$$

is the configuration reached from $\mathbf{q}$ after following constant velocity $\dot{\mathbf{q}}$ during unit time.

Note that the definitions above stem from the Riemanian structure of the configuration space of the robot. The above sum corresponds to the exponential map. One can easily state that "following a constant velocity" makes sense for the four types of joints defined in Table 2.1. We refer the reader to [Absil 2008] Chapter 5 for details about Riemanian geometry.

2.5.3 Straight interpolation

Let $\mathbf{q}_1, \mathbf{q}_2 \in \mathcal{CS}$ be two configurations. Straight interpolation between $\mathbf{q}_1$ and $\mathbf{q}_2$ is defined as the curve in $\mathcal{CS}$ defined on interval $[0,1]$ by:

$$t \to \mathbf{q}_1 + t(\mathbf{q}_2 - \mathbf{q}_1)$$

This interpolation corresponds to the linear interpolation for translation and bounded rotations, to the shortest arc on $\mathbf{S}^1$ for unbounded rotation and to the so called slerp interpolation for $SO(3)$.

Chapter 3

A gradient-based path optimization method for motion planning

Contents

3.1 Motivations

In this chapter, we propose a method aiming at shortening path length after a path planning step. Note that we do not address path planning, but that we take the result of a probabilistic motion planner as the input to our path optimization method.

For this shortening purpose, random shortcut (RS) methods are still very popular [Sekhavat 1998, Geraerts 2007, Hauser 2010]. However, RS requires fine tuning of the termination condition and is not efficient for long trajectories where only a

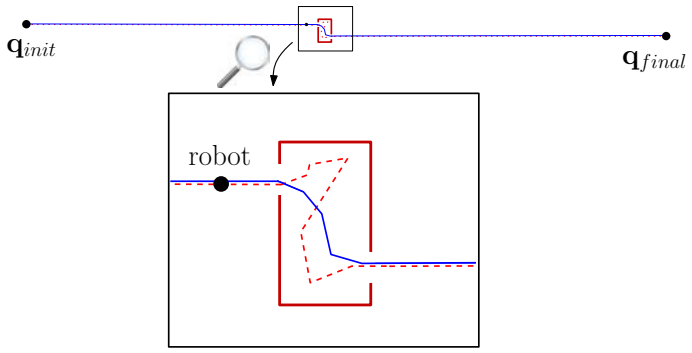


Figure 3.1: Case of a long initial path from $\mathbf{q}_{init}$ to $\mathbf{q}_{final}$ (above) containing a small part that can be optimized (below). Random shortcut is unlikely to optimize the initial dashed part containing detours in the box, whereas our method succeeds (in blue). This type of issue is common in navigation problems, where environments contain long corridors.

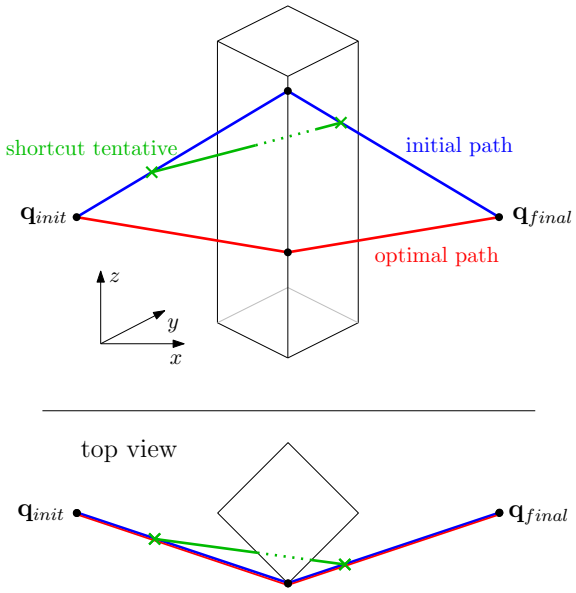
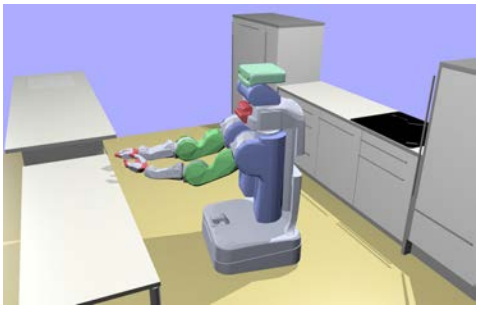


Figure 3.2: Example of a path in $\mathbb{R}^3$. The optimal path belongs to the $x-y$ plane containing $\mathbf{q}_{init}$ and $\mathbf{q}_{final}$. Random shortcut will never manage to optimize the initial path: each shortcut attempt will provide a collision.

minor part needs to be optimized, as in Figure 3.1. Figure 3.2 presents another situation where RS will always fail to optimize the initial path, since it cannot decouple the robot DOFs on which the optimization occurs. This problem is addressed by our method. Processing a path pruning [Geraerts 2007], in order to remove redundant nodes from the initial path, is a classic preliminary step for path length shortening. A pruning will efficiently solve the example introduced in Figure 3.1, however it will fail tackling the issue in Figure 3.2, as RS.

On the other hand, numerical optimization methods like CHOMP [Zucker 2013] can be used as a post-processing step. They have clear termination conditions, but collision avoidance is handled by inequality constraints sampled at many points



	CT (ms)
Collision-checking	8.07
Distance	60.2

Table 3.1: Mean computation times (CT) over 1000 samples for a PR2 robot in a kitchen. The robot and the environment are made of meshes with no geometry reduction. 34 % of the configurations were not collision-free so the collision-checker stopped after finding a collision (which may have reduced the number of checks), contrary to the distances which are computed between all objetcs.

along the trajectory. These methods therefore require a pre-processing step of the robot (and/or environment) model in order to make it simpler: [Zucker 2013] covers PR2 bodies with spheres, while [Schulman 2014] needs to decompose objects into convex subsets. These simplifications are necessary because these methods rely on robot-obstacle distance computation which may be computationally expensive. For instance, Table 3.1 presents a comparison between mean computation times of collision-checking and distance in HPP.

Finally it should be noticed that optimality in robot motion is a notion that should be clarified. Most of the time motion planners provide an optimized motion, which is not optimal at all, but is the output of a given optimization method. When optimal motions exist, numerical algorithms mostly fail in accounting for their combinatorial structure. In addition, optimization algorithms bypass (not overcome) the question of the existence of optimal motions [Laumond 2014]. In that perspective, a path optimization algorithm has to be evaluated with respect to other existing optimization techniques, from qualitative properties and from computational performance.

The idea of our method is to find a good trade-off between the simplicity of *blind* methods like shortcut algorithms, and the complexity of distance based optimization techniques. The method iteratively shortens the initial path with gradient-based information. When a collision is detected at a given iteration, the method backtracks to the latest valid iteration and inserts a one-dimensional constraint between the objects detected in collision. Only collisions between objects are evaluated, therefore no pre-processing of either the robot or environment models is necessary to increase distance computation speed. Respecting the problem geometry also preserves that a solution can still be found, e.g. for narrow passages as holes or grippers. The method is also repeatable since no randomness is introduced. The underlying optimization algorithm is a Linearly Constrained Quadratic Program (LCQP).

Another important feature of our contribution is that we optimize paths on the

robot configuration space in a proper mathematical way. Most other optimization algorithms represent $SO(3)$ rotations by a vector directed along the rotation axis and the norm of which is the rotation angle, also known as the exponential map of $SO(3)$, or even worse by Euler angles.

3.2 Problem definition

This section describes the establishment of the Gradient-based optimizer. The method works as a classic LCQP, reducing the path length expressed as a cost function and avoid collisions with linearized constraints. Details of the LCQP elements will be given in the following subsections. They are associated to functions that will populate the algorithm, presented in the last section.

3.2.1 Optimization variables

We consider as input a collision-free path composed of a concatenation of straight interpolations between $N+2$ configurations: $(\mathbf{q}_0, \mathbf{q}_1, \cdots, \mathbf{q}_{N+1})$. This path is the output of a random sampling path planning algorithm between $\mathbf{q}_0$ and $\mathbf{q}_{N+1}$. We wish to find a sequence of waypoints $\mathbf{q}'_1$,...,$\mathbf{q}'_N$ such that the new path $(\mathbf{q}_0, \mathbf{q}'_1, \cdots, \mathbf{q}'_N, \mathbf{q}_{N+1})$ is shorter and collision-free. Note that $\mathbf{q}_0$ and $\mathbf{q}_{N+1}$ are unchanged. We denote by $\mathbf{x}$ the optimization variable:

$$\mathbf{x} \triangleq (\mathbf{q}_1, \cdots, \mathbf{q}_N)$$

Each path $\mathbf{x}$ is a mapping from interval $[0,1]$ into $\mathcal{CS}$: $\mathbf{x}(0) = q_0$, $\mathbf{x}(1) = q_{N+1}$. Finally, a continuous collision checker inspired of [Schwarzer 2004] is used to validate paths. It also returns the first colliding configuration and its abscissa along the path.

3.2.2 Cost

Let $W \in \mathbb{R}^{p \times p}$ be a diagonal matrix of weights:

$$W = \begin{pmatrix} w_1 I_{p_1} & & 0 & \\ & w_2 I_{p_2} & & \\ & & \ddots & \\ 0 & & & w_m I_{p_m} \end{pmatrix}$$

where I_{p_i} is the identity matrix of size p_i and w_i is the weight associated to the joint J_i. We define the length of the straight interpolation between two configurations as:

$$\|\mathbf{q}_2 - \mathbf{q}_1\|_W \triangleq \sqrt{(\mathbf{q}_2 - \mathbf{q}_1)^T W^2 (\mathbf{q}_2 - \mathbf{q}_1)}$$

Weights are used to homogenize translations and rotations in the velocity vector. For a rotation, the weight is equal to the maximal distance of a point of the body to the center of the joint. For a translation, it is equal to 1.

Given $\mathbf{q}_0$ and $\mathbf{q}_{N+1}$ fixed, the cost we want to minimize is defined by

$$C(\mathbf{x}) \triangleq \frac{1}{2}\sum_{k=1}^{N+1} \lambda_{k-1}\|\mathbf{q}_k - \mathbf{q}_{k-1}\|_W^2$$

The influence of λ_{k-1} coefficients will be commented in Section 3.7. Note that C is not exactly the length of the path, but it can be established that minimal length paths also minimize C. This latter cost is better conditioned for optimization purposes.

The gradient of the cost function $\nabla C(\mathbf{x})$ is computed as follows:

$$\nabla C(\mathbf{x}) = \Big((\lambda_k(\mathbf{q}_{k+1} - \mathbf{q}_k)^T - \lambda_{k+1}(\mathbf{q}_{k+2} - \mathbf{q}_{k+1})^T)W^2\Big)_{k\in\{0..N-1\}}$$

From the gradient expression, we notice that the Hessian H is constant:

$$\mathrm{H} = \begin{pmatrix} (\lambda_0+\lambda_1)W^2 & -\lambda_1 W^2 & 0 & \cdots & & 0 \\ -\lambda_1 W^2 & (\lambda_1+\lambda_2)W^2 & -\lambda_2 W^2 & 0 & \cdots & 0 \\ 0 & -\lambda_2 W^2 & (\lambda_2+\lambda_3)W^2 & -\lambda_3 W^2 & 0 & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -\lambda_{N-2}W^2 & (\lambda_{N-2}+\lambda_{N-1})W^2 & -\lambda_{N-1}W^2 \\ 0 & \cdots & \cdots & 0 & -\lambda_{N-1}W^2 & (\lambda_{N-1}+\lambda_N)W^2 \end{pmatrix}$$

3.3 Unconstrained resolution

We assume that the direct interpolation between the initial and final configurations contains collisions. An iteration at stage i is described as follow:

$$\begin{aligned} \mathbf{p}_i &= -\mathrm{H}^{-1}\nabla C(\mathbf{x}_i)^T \\ \mathbf{x}_{i+1} &= \mathbf{x}_i + \alpha_i \mathbf{p}_i \end{aligned} \tag{3.1}$$

where α_i is a real-valued parameter. Taking $\alpha_i = 1$ yields the unconstrained minimal cost path, i.e. all waypoints aligned on the straight line between $\mathbf{q}_0$ and $\mathbf{q}_{N+1}$. Since this solution is in collision, we set $\alpha_i = \alpha_{init}$ where α_{init} is a parameter in interval $[0,1]$.

Computation of $\mathbf{p}_i$ from Equation (3.1) is associated to an unconstrained version of function COMPUTEITERATE.

We iterate step (3.1) until path $\mathbf{x}_{i+1}$ is in collision. When a collision is detected, we introduce a constraint and perform a new iteration from $\mathbf{x}_i$ as explained in the next section.

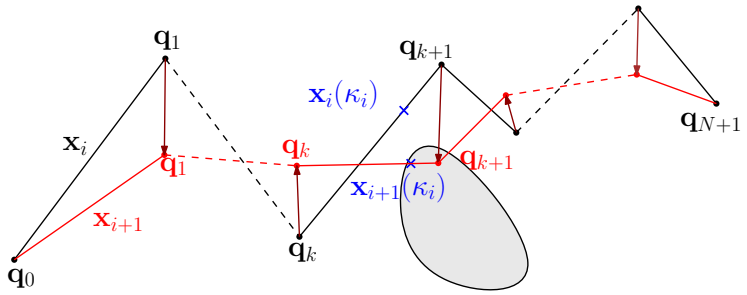


Figure 3.3: Illustration of one iteration of the path optimization. $\mathbf{x}_{i+1}$ appears to be in collision with the obstacle. The first colliding configuration $\mathbf{x}_{i+1}(\kappa_i)$ at abscissa κ_i is returned by the continuous collision checker. The corresponding constraint will be computed in the backtracked configuration $\mathbf{x}_i(\kappa_i)$.

3.4 Linear constraints

Let us assume that at iteration i, j linear constraints have been inserted before the current iteration. These constraints are stored as lines of a matrix as follows:

$$\Phi_i = \begin{pmatrix} L_1 \\ \vdots \\ L_j \end{pmatrix}$$

where the step $\mathbf{p}_i$ is constrained to be in the kernel of Φ_i as follows:

$$\Phi_i \, \mathbf{p}_i = 0$$

These linear constraints are built from the linearization of a collision-constraint function, which will be detailed in the following section.

3.4.1 New constraint

As illustrated in Figure 3.3, let us denote by κ_i the abscissa of the first collision detected on path $\mathbf{x}_{i+1}$, which previous iteration $\mathbf{x}_i$ was collision-free. Thus in configuration $\mathbf{x}_{i+1}(\kappa_i)$ a collision has been detected. Two cases are possible:

1. The collision occurred between two bodies of the robot: $\mathcal{B}_1$ and $\mathcal{B}_2$.

2. The collision occurred between a body of the robot $\mathcal{B}_1$ and the environment.

In the rest of this section, the first case only will be considered. Reasoning about the second case is similar, except that the constraint is on the position of $\mathcal{B}_1$ with respect to the environment.

The principle of the method is to compute a linear constraint, initialized on the collision-free configuration $\mathbf{x}_i(\kappa_i)$ to avoid the collision appearing at $\mathbf{x}_{i+1}(\kappa_i)$. To handle this, we introduce a one-dimensional constraint based on the orthogonal direction of the encountered collision.

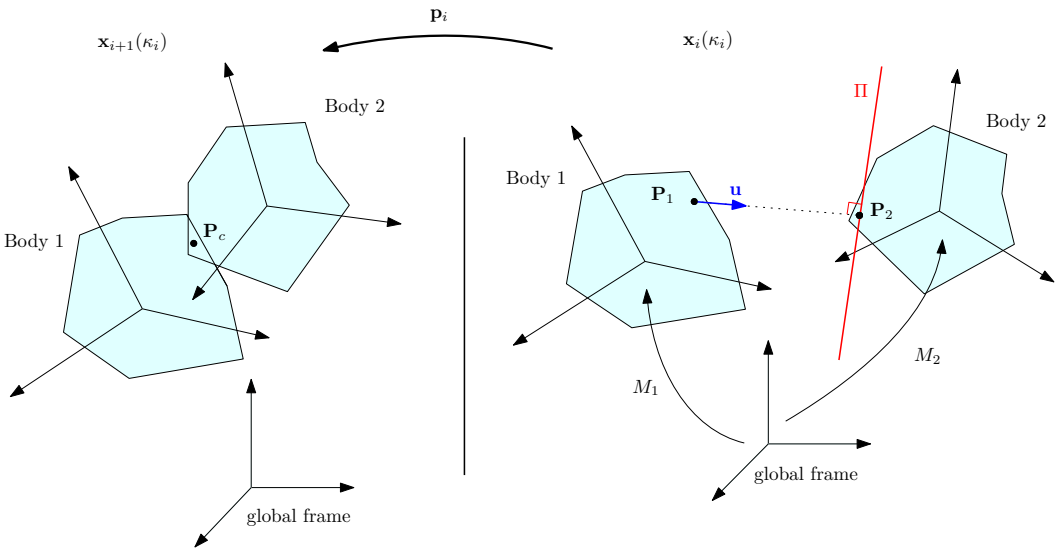


Figure 3.4: Bodies representation in collision-configuration (left) and backtracked collision-free configuration (right). A contact point (i.e. any point in the intersection of the bodies) $\mathbf{P}_c$ can be returned by the Flexible Collision Library [Pan 2012a], used to detect collisions. Constraint defined by Equation (3.4) aims at keeping $\mathbf{P}_2(\mathbf{q})$ in plane Π fixed to $\mathcal{B}_1$.

At the collision-configuration $\mathbf{x}_{i+1}(\kappa_i)$, let $\mathbf{P}_c \in \mathbb{R}^3$ be a contact point expressed in the global frame (Figure 3.4 left). We denote by:

- $\mathbf{P}_1^{loc}$ (resp. $\mathbf{P}_2^{loc}$) the coordinate vector of $\mathbf{P}_c$ in the local frame of $\mathcal{B}_1$ (resp. $\mathcal{B}_2$).
- $M_1(\mathbf{q}) \in SE(3)$ (resp. $M_2(\mathbf{q}) \in SE(3)$) the rigid-body transformation representing the position of $\mathcal{B}_1$ (resp. $\mathcal{B}_2$) in the global frame, in configuration $\mathbf{q}$.
- $M_2^1(\mathbf{q}) = M_1(\mathbf{q})^{-1} M_2(\mathbf{q})$ the position of $\mathcal{B}_2$ local frame in $\mathcal{B}_1$ local frame, in configuration $\mathbf{q}$.
- $\mathbf{P}_1(\mathbf{q})$ (resp. $\mathbf{P}_2(\mathbf{q})$) the points moving with $\mathcal{B}_1$ (resp. $\mathcal{B}_2$) of local coordinate $\mathbf{P}_1^{loc}$ (resp. $\mathbf{P}_2^{loc}$) in $\mathcal{B}_1$ (resp. $\mathcal{B}_2$) local frame.

We define $\mathbf{u}$ as the coordinate vector of the unit vector linking points $\mathbf{P}_1$ and $\mathbf{P}_2$ in configuration $\mathbf{x}_i(\kappa_i)$, expressed in local frame of $\mathcal{B}_1$ (Figure 3.4 right):

$$\mathbf{u} = \frac{M_2^1(\mathbf{x}_i(\kappa_i))\mathbf{P}_2^{loc} - \mathbf{P}_1^{loc}}{\|M_2^1(\mathbf{x}_i(\kappa_i))\mathbf{P}_2^{loc} - \mathbf{P}_1^{loc}\|}$$

Note that $\mathbf{u}$ is well defined since configuration $\mathbf{x}_i(\kappa_i)$ is collision-free.

Let g be the real valued function mapping the projection of vector $\mathbf{P}_1\mathbf{P}_2(\mathbf{q})$ on $\mathbf{u}$ to a configuration $\mathbf{q}$. For any $\mathbf{q} \in \mathcal{CS}$:

$$g(\mathbf{q}) = \left(M_2^1(\mathbf{q})\mathbf{P}_2^{loc} - \mathbf{P}_1^{loc} | \mathbf{u}\right) \tag{3.2}$$

Let f be the function defined from $\mathcal{CS}^N$ to $\mathbb{R}$ by:

$$f(\mathbf{x}) = g(\mathbf{x}(\kappa_i)) \tag{3.3}$$

The constraint defined for any path $\mathbf{x}$ by:

$$f(\mathbf{x}) - f(\mathbf{x}_i) = 0. \tag{3.4}$$

aims at keeping point $\mathbf{P}_2(\mathbf{q})$ in a plane attached to $\mathcal{B}_1$, orthogonal to $\mathbf{u}$ and and passing by $\mathbf{P}_2$ in configuration $\mathbf{x}_i(\kappa_i)$ (Figure 3.4 right).

We linearize the constraint around $\mathbf{x}_i$:

$$\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_i)(\mathbf{x} - \mathbf{x}_i) = 0$$

The computation of the linearized constraint is described in the next section. Then, a line is added in the constraint Jacobian matrix Φ_i:

$$\Phi_{i+1} = \begin{pmatrix} L_1 \\ \vdots \\ L_{j+1} \end{pmatrix} \text{ with}$$
$$L_{j+1} = \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_i)$$

This stage is performed by function addCollisionConstraint.

Finally, we refer to [Nocedal 2006] for solving LCQP. The step computation is associated to a constrained version of computeIterate.

3.4.2 Linearized constraint computation

Let $\mathbf{q}_{k,i}$ denote the waypoint k along path $\mathbf{x}_i$. There exist $\beta \in [0,1]$ and k such that $\mathbf{x}_i(\kappa_i)$ can be written as a combination of two waypoints:

$$\mathbf{x}_i(\kappa_i) = \mathbf{q}_{k,i} + \beta(\mathbf{q}_{k+1,i} - \mathbf{q}_{k,i})$$

Thus the linearized constraint Jacobian $\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_i)$ is built by matrix blocks using the Jacobian $\frac{\partial g}{\partial \mathbf{q}}$ expressed in each of the two waypoints and β. This step is performed by computeCollisionConstraint.

3.5 Convergence analysis and algorithm refinement

Although linearized constraints may differ from the initial geometrically relevant non-linear constraint when the iterate goes away from the linearization path, we show in this section that our algorithm converges under some reasonable assumptions. The underlying idea of the proof is sketched in an analogous problem to the path shortening in Figure 3.5.

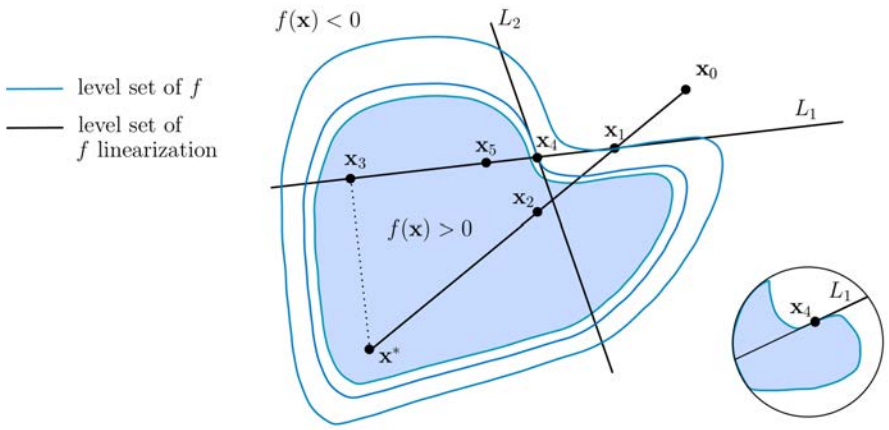


Figure 3.5: Illustration of linearly constrained quadratic program on an analogous problem to the path shortening problem. Iterations $\mathbf{x}_i$ are represented for $\alpha_{init} = 0.25$. The non-linear constraint is defined as $f(\mathbf{x}) \leq 0$ and the linearized ones as the L_i lines. The bottom right picture shows a condition for the linearized constraint L_1 to be linearly dependent on L_2 by being stationary at the boundary of f.

In this problem, the quadratic cost $\frac{1}{2}\|\mathbf{x}-\mathbf{x}^*\|^2, \mathbf{x} \in \mathbb{R}^2$ is minimized under the non-linear constraint $f(\mathbf{x}) \leq 0$. The algorithm starts from $\mathbf{x}_0$. The first iterate is $\mathbf{x}_1$ which satisfies the constraint. The second iterate is $\mathbf{x}_2$ that does not satisfy the constraint. The algorithm backtracks to $\mathbf{x}_1$ and inserts linear constraint $L_1 : \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_1)(\mathbf{x}-\mathbf{x}_1) = 0$. $\mathbf{x}_3$ is the global minimum under L_1. As $\mathbf{x}_3$ does not satisfy the constraint, the algorithm moves to $\mathbf{x}_4$ that satisfies the constraint, and then to $\mathbf{x}_5$ that does not. The algorithm backtracks to $\mathbf{x}_4$, inserts constraint $L_2 : \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_2)(\mathbf{x}-\mathbf{x}_2) = 0$, and returns $\mathbf{x}_4$ as a solution since the dimension of the search space is 0. Notice that the same non-linear constraint may give rise to several linear constraints. The convergence of the algorithm relies on the fact that the kernel of constraint L_2 is not contained in the kernel of the current constraints (L_1 only here). The convergence analysis can be roughly summarized as follows. L_2 to be linearly dependent of L_1 requires that f is stationary along L_1 at $\mathbf{x}_4$. This is unlikely (but possible) since $\mathbf{x}_4$ is not far from the boundary of the domain defined by $f(\mathbf{x}) \leq 0$. If L_2 was not linearly independent from L_1, the algorithm would keep searching new iterates between $\mathbf{x}_5$ and $\mathbf{x}_4$. If by any chance constraint f linearized around each of those collision-free iterates was each time linearly dependent from L_1 the iterates would converge to the boundary of the domain defined by $f(\mathbf{x}) \leq 0$. By continuous differentiability of f, this would mean that f is stationary at the boundary. In other words, the straight line passing by $\mathbf{x}_1$ and $\mathbf{x}_3$ would cross the boundary tangentially (as in Figure 3.5 bottom right picture). This is possible but unlikely, unless the problem has been defined as such on purpose.

We now clarify on what assumption the constraints are linearly independent, similarly to the analogous problem. From the definition of Φ_i, it is straightforward that:

$$\text{Ker } \Phi_{i+1} \subset \text{Ker } \Phi_i \tag{3.5}$$

In other words, any path iteration complying with the set of constraints contained in Φ_{i+1} will satisfy the set Φ_i. Let us assume that:

$$L_{j+1}\mathbf{p}_i \neq 0 \tag{3.6}$$

We will elaborate later on this assumption. This means that $\mathbf{p}_i \notin \text{Ker } \Phi_{i+1}$, and as $\mathbf{p}_i \in \text{Ker } \Phi_i$, then:

$$\text{Ker } \Phi_i \neq \text{Ker } \Phi_{i+1}$$

From Equation (3.5), we deduce that:

$$\dim(\text{Ker } \Phi_{i+1}) < \dim(\text{Ker } \Phi_i)$$

This result proves that under Assumption (3.6), each additional constraint is linearly independent from the previous ones. Thus, the dimension of search space decreases and our algorithm terminates in a finite number of iterations.

3.5.1 Geometrical representation of the dependency between linear constraints

As in the previous section, we assume that $\mathbf{x}_i$ is collision-free and that $\mathbf{x}_{i+1}$ is in collision at abscissa κ_i. According to Equation (3.3), the evaluation of the constraint function f along the iteration line $\mathbf{x}_i + t\alpha_i\mathbf{p}_i,\ t \in [0,1]$ going from $\mathbf{x}_i$ to $\mathbf{x}_{i+1}$ can be written as follows:

$$f(\mathbf{x}_i + t\alpha_i\mathbf{p}_i) = g\left((\mathbf{x}_i + t\alpha_i\mathbf{p}_i)(\kappa_i)\right) \tag{3.7}$$

The argument of function g above is a trajectory in the robot configuration space that we denote by Γ:

$$\Gamma(t) = (\mathbf{x}_i + t\,\alpha_i\mathbf{p}_i)(\kappa_i),\ t \in [0,1] \tag{3.8}$$

The trajectory Γ is defined by taking the constant abscissa κ_i and by moving from path $\mathbf{x}_i$ along step $\mathbf{p}_i$ (see Figure 3.6). Note that configuration $\mathbf{x}_{i+1}(\kappa_i)$ is reached when t is equal to 1. Substituting Equation (3.8) into Equation (3.7) and differentiating with respect to t yields

$$f(\mathbf{x}_i + t\alpha_i\mathbf{p}_i) = g(\Gamma(t)) \tag{3.9}$$

$$\alpha_i \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_i + t\alpha_i\mathbf{p}_i)\mathbf{p}_i = \frac{d}{dt}g(\Gamma(t)) \tag{3.10}$$

Property 1. *From the definition of g in Equation (3.2), the right hand side of Equation (3.10) represents the velocity of point $\mathbf{P}_2$ in reference frame $\mathcal{B}_1$ projected on vector $\mathbf{u}$ along trajectory Γ.*

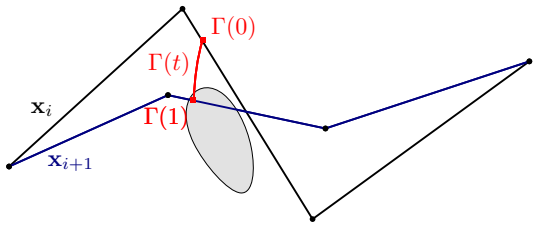


Figure 3.6: Representation in the robot configuration space of the trajectory Γ, defined in Equation (3.8).

Therefore the following expressions

$$\alpha_i \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_i)\mathbf{p}_i = \frac{d}{dt} g(\Gamma(0)) \qquad \alpha_i \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_{i+1})\mathbf{p}_i = \frac{d}{dt} g(\Gamma(1))$$

correspond to $(\mathbf{v}_{\mathbf{P}_2/\mathcal{B}_1}|\mathbf{u})$, respectively in configurations $\mathbf{x}_i(\kappa_i)$ and $\mathbf{x}_{i+1}(\kappa_i)$, where $\mathbf{v}_{\mathbf{P}_2/\mathcal{B}_1}$ represents the velocity of point $\mathbf{P}_2$ in reference frame $\mathcal{B}_1$. Note that Assumption (3.6) is equivalent for the first above equality to be different from 0. In conclusion, Assumption (3.6) is violated (i.e. constraints L_j and L_{j+1} are linearly dependent) if and only if $\mathbf{v}_{\mathbf{P}_2/\mathcal{B}_1}$ is orthogonal to $\mathbf{u}$. Although very unlikely, this case might appear for some new constraint. In this case, inserting constraint L_{j+1} is useless since

$$\text{Ker } \Phi_i = \text{Ker } \Phi_{i+1}$$

3.5.2 Algorithm refinement

When the new constraint is not linearly independent from the set of previous constraints, the algorithm enters an additional loop, performed by Algorithm 1, in order to find a new constraint that is linearly independent. The loop keeps looking for paths along line segment $[\mathbf{x}_i, \mathbf{x}_{i+1}]$ by dichotomy. A pair containing the latest free path and the latest path in collision, denoted by $(\mathbf{x}_{Free}, \mathbf{x}_{Coll})$ is stored along the loop. New iterations are chosen in the middle of this pair.

- If the new path is collision-free, it replaces $\mathbf{x}_{Free}$ in the pair.
- If the new path is in collision, it replaces $\mathbf{x}_{Coll}$ in the pair.

In both cases, a new constraint is built following the method described in Section 3.4.1. Then two cases are possible:

1. at some point in the loop the new constraint is linearly independent from the previous constraints. The new constraint is added to Φ_i to give rise to Φ_{i+1}, and the loop is interrupted, or
2. each new constraint is linearly dependent from the previous constraints and the loop never ends.

In the second case, the iterations of the loop converge to a path that we denote by $\bar{\mathbf{x}}$. $\mathbf{x}_{Free}$ and $\mathbf{x}_{Coll}$ also both converge to $\bar{\mathbf{x}}$. $\bar{\mathbf{x}}$ necessarily lies at the boundary

Algorithm 1 Description of FINDNEWCONSTRAINT() which returns a linearly independent constraint w.r.t. previous constraints stacked in Φ.

Input: $(\mathbf{x}_{Free}, \mathbf{x}_{Coll})$ latest collision-free and in collision paths, $\mathbf{p}$ and α such that $\mathbf{x}_{Coll} \leftarrow \mathbf{x}_{Free} + \alpha\,\mathbf{p}$
Output: linearized constraint $\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_{Free})$
Require: constraint $\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_{Free})$ built from $\mathbf{x}_{Coll}$ would produce a rank loss in constraint Jacobian Φ

```
solved ← false
while (not(solved)) do
    α ← (1/2) α
    x ← x_Free + α p
    if (validatePath(x)) then
        x_Free ← x
    else
        x_Coll ← x
    ∂f/∂x(x_Free) ← computeCollisionConstraint(x_Coll, x_Free)
    solved ← isFullRank(Φ , ∂f/∂x(x_Free))
```

between free paths and paths in collision. Let us denote by $\bar{\kappa}$ the abscissa along $\bar{\mathbf{x}}$ where $\mathcal{B}_1$ and $\mathcal{B}_2$ come to contact, and let us denote by $\bar{\mathbf{P}}$ the contact point.

At each iteration, the new linear constraint

$$L_{j+1} = \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_{Free})$$

is tested. As iterations $\mathbf{x}_{Free}$ and $\mathbf{x}_{Coll}$ tend toward $\bar{\mathbf{x}}$, it can be established by a geometric reasoning analogous to Property 1, that $L_{j+1}\mathbf{p}_i$ tends to the norm of the velocity of point $\bar{\mathbf{P}}$ belonging to $\mathcal{B}_2$ in the frame of $\mathcal{B}_1$. The following property summarizes the result of this section.

Property 2. *As long as along iteration $\mathbf{p}_i$, the trajectory $(\mathbf{x}_i + t\alpha_i\mathbf{p}_i)(\bar{\kappa})$ does not enter in collision with contact point velocity equal to 0 in the frame of $\mathcal{B}_1$, Algorithm 1 converges in a finite number of steps.*

Property 2 means that the gradient-based algorithm converges in all cases, except for ill-defined problems.

3.6 Algorithm

The gradient-based path-optimizer is described in Algorithm 2. Note that the LCQP optimal step $\mathbf{p}$, computed by COMPUTEITERATE, is known. The collision detection on a path is handled by VALIDATEPATH, which returns *true* if the given path is collision-free.

The idea of the algorithm is to process iterations that reduce the path length according to the LCQP cost. If collision-constraints have been added to the LCQP,

Algorithm 2 Gradient-based (GB) algorithm for path-optimization.

Input: path to optimize $\mathbf{x}_0$
Output: optimized collision-free path $\mathbf{x}_0$

```
α ← α_init
minReached ← false
while (not(noCollision and minReached)) do
    p = computeIterate()
    minReached = (||p|| < 10^-3 or α = 1)
    x_1 ← x_0 + α p
    if (not(validatePath(x_1))) then
        noCollision ← false
        if (α ≠ 1) then
            computeCollisionConstraint(x_1, x_0)
            findNewConstraint()
            addCollisionConstraint()
            α ← 1
        else
            α ← α_init
    else
        x_0 ← x_1
        noCollision ← true
return x_0
```

further iterations will comply with them. The algorithm stops when the LCQP minimum is reached and collision-free.

One main difficulty is to handle the scalar parameter α determining how much of the computed step will be traveled along. As presented in Algorithm 2, α takes two values, $\alpha_{init} < 1$ to process small steps, or 1 to go directly to the optimum under the latest set of constraints. This latter case is interesting since, if this optimal path is collision-free, the algorithm has converged and returns the path as the solution. Choosing to travel small steps from a valid path decreases the chances of being in collision. Besides if a collision occurs, the collision-constraint is computed on the last valid path, which is not too much deformed compared to the path that has collisions. As a result, collision-constraints are only computed (computeCollisionConstraint) and added (addCollisionConstraint) when performing a reduced iteration (i.e. $\alpha = \alpha_{init}$).

Note that even though the constraints are linearized, the algorithm converges.

3.7 Results

This part gathers optimization results performed on HPP. Initial paths are obtained with two kinds of probabilistic planners: Visibility-PRM [Siméon 2000] and RRT-connect [Kuffner 2000]. We denote them by PRM and RRT respectively. Unless

another value is provided, α_{init} is set to 0.2. A further section provides a discussion on the α_{init} value tuning.

3.7.1 From 2D basic examples

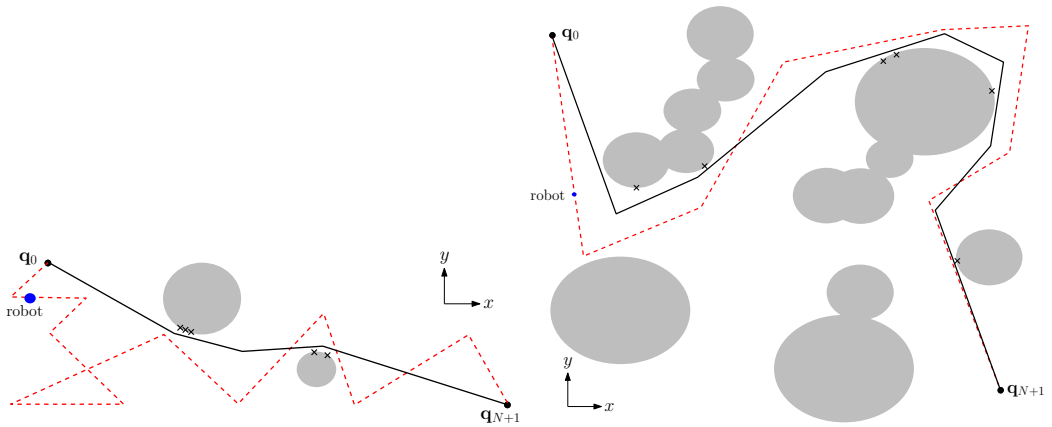


Figure 3.7: Path-optimization results on 2D robots, moving around gray obstacles. Initial paths are dashed and crosses represent contact points $\mathbf{P}_c$ related to collision-constraints. Note that, on the left, the detour completely disappears.

Figure 3.7 shows the result of our optimizer on 2D cases. Contact points which have led to constraints are represented. They permit to understand how the path is kept out of the obstacles while reducing detours. Note that, since not obstacle clearance is considered, the robot may pass close to obstacles.

Figure 3.1 illustrates a *very long* path example which RS or PRS will not manage to optimize in an affordable time, because of probabilistically failing to sample configurations in the box. The GB method succeeds to optimize the path contained in the box, with the following cost coefficients:

$$\lambda_{k-1} = \frac{1}{\sqrt{(\mathbf{q}_{k,0} - \mathbf{q}_{k-1,0})^T W^2 (\mathbf{q}_{k,0} - \mathbf{q}_{k-1,0})}},\ k \in \{1..N+1\}$$

aiming at keeping the same ratio between path segment lengths at minimum as at initial path, represented by the waypoints $(\mathbf{q}_{k,0})_{k\in\{0..N\}}$. Without these coefficients, the path that minimizes the cost corresponds to a straight line with the waypoints equidistantly allocated. This is not relevant for Figure 3.1 type of problems where a local passage is very constrained by obstacles. Note that this cost is also working with all other examples presented in this section, and provides better quality results than the original cost.

3.7.2 To 3D complex problems

3.7.2.1 Comparison to random shortcut algorithms

Algorithm 3 Random shortcut as adapted from [Sekhavat 1998] Section 6.4.1. STRAIGHTINTERPOLATION returns the linear interpolation between two configurations. $\mathbf{x}_{|I}$ denotes path $\mathbf{x}$ restricted to interval I. t_{lim} represents the duration limitation of the algorithm.

Input: path to optimize $\mathbf{x}$, time limit t_{lim}
Output: optimized collision-free path $\mathbf{x}$

$t_{start} \leftarrow$ CURRENTTIME()
$t \leftarrow 0$
while $t < t_{lim}$ **do**
  $failure \leftarrow true$
  $t_1 < t_2 \leftarrow$ random numbers in $[0, 1]$
  $lp0 \leftarrow$ STRAIGHTINTERPOLATION$(\mathbf{x}(0), \mathbf{x}(t_1))$
  $lp1 \leftarrow$ STRAIGHTINTERPOLATION$(\mathbf{x}(t_1), \mathbf{x}(t_2))$
  $lp2 \leftarrow$ STRAIGHTINTERPOLATION$(\mathbf{x}(t_2), \mathbf{x}(1))$
  $newPath \leftarrow$ empty path defined on $[0, 0]$
  if VALIDATEPATH$(lp0)$ **then**
    $newPath \leftarrow lp0$;
  else
    $newPath \leftarrow \mathbf{x}_{|[0,t_1]}$
  if VALIDATEPATH$(lp1)$ **then**
    $newPath \leftarrow$ CONCATENATE$(newPath, lp1)$
  else
    $newPath \leftarrow$ CONCATENATE$(newPath, \mathbf{x}_{|[t_1,t_2]})$
  if VALIDATEPATH$(lp2)$ **then**
    $newPath \leftarrow$ CONCATENATE$(newPath, lp2)$
  else
    $newPath \leftarrow$ CONCATENATE$(newPath, \mathbf{x}_{|[t_2,1]})$
  $\mathbf{x} \leftarrow newPath$
  $t \leftarrow$ CURRENTTIME$() - t_{start}$
return x

Our algorithm has also been experimented on more complex robots and environments[1]. In Figures 3.8, 3.9, 3.12 and 3.13, we present multiple situations where the GB algorithm is tested and compared to RS and PRS. After describing the random optimizers, we will present each benchmark and its qualitative path results. Then quantitative convergence graphs and averages will be given and discussed.

The RS implementation is given in Algorithm 3. RS shortens the path by randomly sampling configurations along it, and by trying to link them with collision-free interpolations. The termination condition of RS is a duration time limit t_{lim},

[1] Video of the experimental results is available at https://youtu.be/1MFn0en51qI

and is typically set as the GB convergence time. Concerning PRS, its implementation is identical to [Geraerts 2007]: for a random DOF, configurations are sampled along the initial path as in Algorithm 3. The straight interpolation returns a path made of an interpolation only on the current DOF, while it is based on the previous subpath for other DOFs. If this path is collision-free, it is added to the final path as in RS. The process is stopped when the duration exceeds t_{lim}.

Before entering the manipulator examples, the GB algorithm is analyzed on a popular problem in the motion planning literature: a freeflyer puzzle, corresponding to Figure 3.12(b). The puzzle has to cross down the obstacle using the hole in the middle. The initial path planned with PRM contains detours above and below the obstacle, as well as small superfluous motions in the hole. Results of the three optimizers are similar in terms of path length. Note that for GB, trajectory parts above and below the obstacle are not completely shorten, i.e. the puzzle center is still committing detours. This is the result of adding collision-constraints on these parts of the trajectory, between one of the puzzle branches and the obstacle. In total, 43 collision-constraints have been produced. One idea could be to arbitrarily cancel constraints in these upper and lower parts of the trajectory, and to keep the ones in the hole. However we want the present GB algorithm to remain general and basic, such constraint relaxation is part of the possible future work.

In the double-arms benchmark (4 DOFs), Figure 3.12(a), one arm has to get around a cylinder obstacle while the other arm stays in the same configuration. As expected, the initial path given by RRT activates both arms to solve the problem. Unlike RS, the GB optimizer manages to cancel the rotations of the free arm while optimizing the first arm motion, creating collision-constraints with the cylinder obstacle.

Some problems involve a 6-axis manipulator arm, also called UR5, equipped with a bar or a gripper. In a relatively free environment, represented in Figure 3.12(c), results from our method and RS are similar. Note also that the end-effector trajectory is completely different from the initial one: the robot is easily passing between the meshed spheres, keeping its end-effector above. For an UR5 working in a cluttered environment inspired by an industrial issue (Figure 3.8), GB path optimization efficiently returns a shorter solution, close to the result of RS and to what can be observed in reality.

A problem involving a Baxter-like[2] robot manipulating in an office environment is presented in Figure 3.13(c). The robot starts with its end-effectors above the computer and has to turn and reach the shelf. According to the quality of the left-wrist trajectories, the GB optimizer provides the smoothest motion.

In the three following high-DOF examples involving the PR2 robot (35 DOFs), note that results are better in terms of path quality, as a result of the parasite DOF motion removal.

In the example shown in Figure 3.9, PR2 simply has to cross its arms from the left arm up position to the right arm up one, without any assumption about

[2] A torso rotation was added and the grippers were removed.

the group of DOFs to activate (i.e. no DOF is locked). The RRT planner returns detours and activates non-useful DOFs such as the head, the torso lift and the mobile base. Such behavior induces a high initial path length. RS hardly optimizes the mobile base translation (Figure 3.9 middle) of the robot and other unnecessary DOF uses. Whereas the GB optimized-path mainly results in moving the arms as expected (Figure 3.9 right), just creating two collision-constraints between the arms.

In the PRS result, only presented in the video, the motion is less optimized than with RS: the arms are moving widely and the mobile base remains activated. One solution to remove such unnecessary DOF activation, can be to try applying a partial shortcut on each DOF between the initial and final configurations. It appears that this step is more costly in terms of computation time than the GB duration, therefore it cannot be afforded by our PRS implementation, due to the t_{lim} condition. However, this solution could be applied as a preliminary optimization stage for each optimizer.

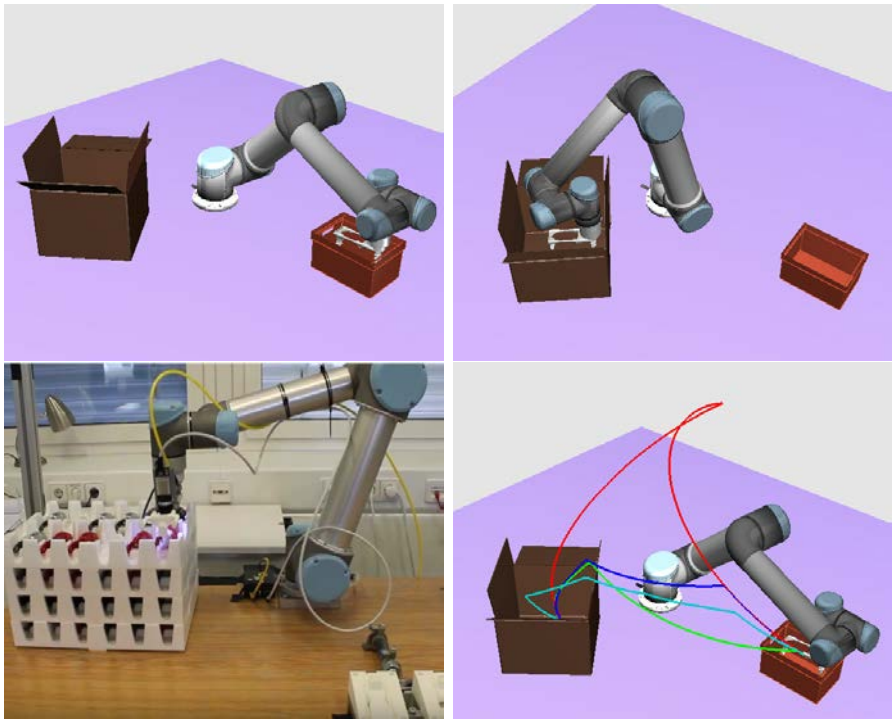


Figure 3.8: (Bottom left) An industrial use-case example proposed by Philips for the Factory-in-a-Day project[3]. A similar environment has been created (top) to illustrate that our method can comply with an industrial problem, where initial and final configurations of the UR5 are constrained in boxes. End-effector trajectories are illustrated (bottom right): RRT planning in red, a RS optimization in blue, a PRS one in cyan and the GB optimization in green.

Similar results are obtained on the PR2 performing manipulation tasks in a

[3]Source: Robothon of Factory In A Day - Philips case. Video: https://youtu.be/fhKlfVsupOE

Problem		Computation time	Relative remaining length (%) GB	RS	PRS
Freeflyer-puzzle	($\alpha_{init} = 0.05$)	742 ms	53.0	41.4	46.1
Double-arms	(RRT)	29.0 ms	44.7	53.6	56.6
UR5-with-spheres	(RRT, $\alpha_{init} = 0.3$)	453 ms	48.5	42.1	72.0
UR5-industrial-example		765 ms	40.3	29.6	43.4
Baxter-in-office		18.8 s	36.5	45.2	79.8
PR2-crossing-arms		882 ms	19.9	43.2	95.2
PR2-in-kitchen-1		13.5 s	28.3	42.7	90.6

Table 3.2: Average results for 50 runs of several examples. For each run, a solution path is planned by Visibility-PRM (unless 'RRT' for RRT-connect is mentioned) as initial guess for the three optimizers. RS and PRS results correspond to averages of 50 launches of the random optimizers on each initial guess. The GB computation time is the work duration allowed for the random optimizers. $\alpha_{init} = 0.2$ unless another value is specified. Boxes highlight the best path length reduction result among the three optimizers.

kitchen environment. Firstly, the robot moves its hands from the top to the bottom of a table. The different trajectories of the right gripper are indicated in Figure 3.13(a). Our optimizer manages to reduce the initial path length from PRM and improves the path quality just adding constraints between the table and the robot arms. Thus, the robot just slightly moves backward and uses its arm DOF to avoid the table, instead of processing a large motion to move away from the table. Secondly, another example of PR2 going from the set to the fridge door is presented in Figure 3.13(b) with the mobile base trajectories. Here, GB and RS results are similar in terms of length and rendering.

For some of the presented benchmarks, convergence graphs of the path length reduction are given in Figure 3.14. The chosen initial paths are unchanged, i.e. correspond to Figures 3.12 and 3.13. Each graph illustrates the percent ratio of the optimized path length over the initial path length, during the optimization. It is not a surprise that GB is globally slower than RS due to the difference of the computations complexity during the optimization. Thus RS converges faster. However, it seems that GB catches up and overcomes RS before ending (see Figures 3.14(d) and 3.14(e)), thanks to the optimization of the mobile base motion. Therefore, it could be interesting to investigate the performance of a composed optimizer, starting by a RS stage until *convergence* and finishing by a GB stage to improve the path length reduction. In the puzzle example (see Figure 3.14(b)), the difference of optimization speed between GB and the random optimizers is significant. This can be partly explained by the fact that collision checking is rapidly performed in such basic geometry problem. This favors the random shortcut tries while GB spends time on the LCQP resolution.

Since the GB optimizer results depend on the shape of the initial guess, e.g. the number of waypoints and the proximity to obstacles, results averages for 50 initial paths of each benchmark are presented in Table 3.2. As mentioned, the paths are obtained from PRM or RRT. Due to their nature, these motion planners provide

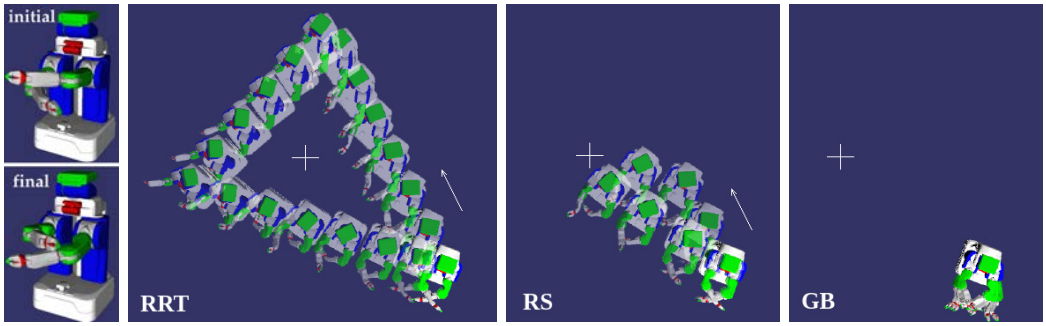


Figure 3.9: PR2-crossing-arms example: the PR2 robot has just to exchange the positions of its arms (left). The task is simple, however, in absence of explicit indication, any probabilistic motion planner will compute a path that makes the PR2 mobile base purposelessly move around the + marker. Path optimization is expected to remove unnecessary motions. RS fails in this case while GB succeeds

different types of path: the output of PRM contains less waypoints and does not tend to be close to the contact, behavior induced by the extension process of RRT.

In some cases, α_{init} is reduced to comply with very narrow passages, or increased in the opposite case.

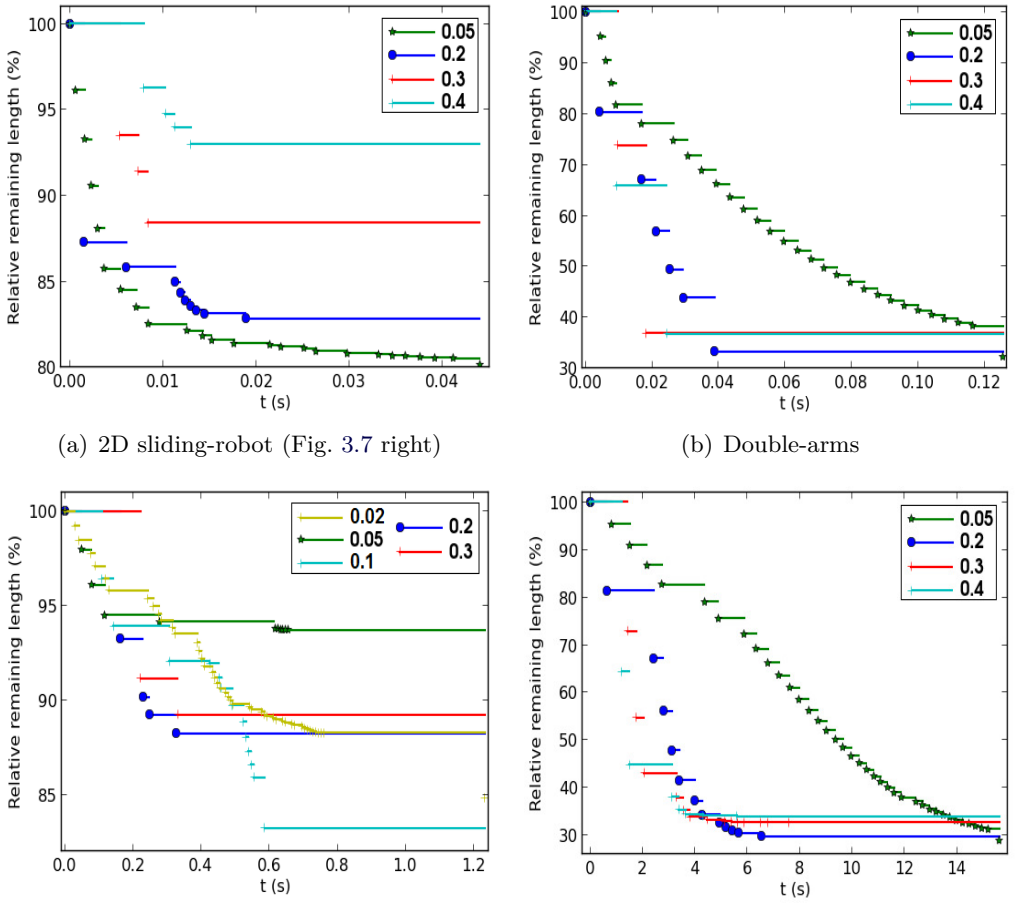
The results seem to be consistent with the trajectory analysis and convergence graphs. Except the low-DOF problem of the puzzle and the UR5, our method provides shorter or similar results compared to RS. Results even seem to be better when the number of DOFs increases, as the baxter and PR2 examples.

3.7.2.2 Analysis of α_{init} influence

This section deals with the influence of the parameter α_{init} on the GB convergence. Reducing α_{init} makes Algorithm 2 process smaller iterations. Some expected behaviors are visible in Figure 3.10. For instance, Figure 3.10(a) illustrates an expected influence of a α_{init} reduction on the final path lengths. Besides, commonly to Figures 3.10(a) , 3.10(b) and 3.10(d), $\alpha_{init} = 0.05$ has the higher convergence time.

However, due to the strong non-linearity of the constraints, reduced iterations do not necessarily lead to a slower but refined solution. GB can stop earlier in local minimum, which may also have a better path length reduction. This is the case of Figure 3.10(b) where $\alpha_{init} = 0.2$ results in a shorter path than $\alpha_{init} = 0.05$. More surprisingly in Figure 3.10(b), $\alpha_{init} = 0.05$ yields the worse reduction.

Instead of investigating a way to find a constant α_{init} conditioned by the problem and the initial path, we plan to adapt α_{init} during the optimization process. This can be achieved by taking into account geometrical considerations inspired from the continuous collision checker. For instance, the collision checker is able to return a lower bound of the distance between objects.



(a) 2D sliding-robot (Fig. 3.7 right)

(b) Double-arms

(c) Freeflyer-puzzle

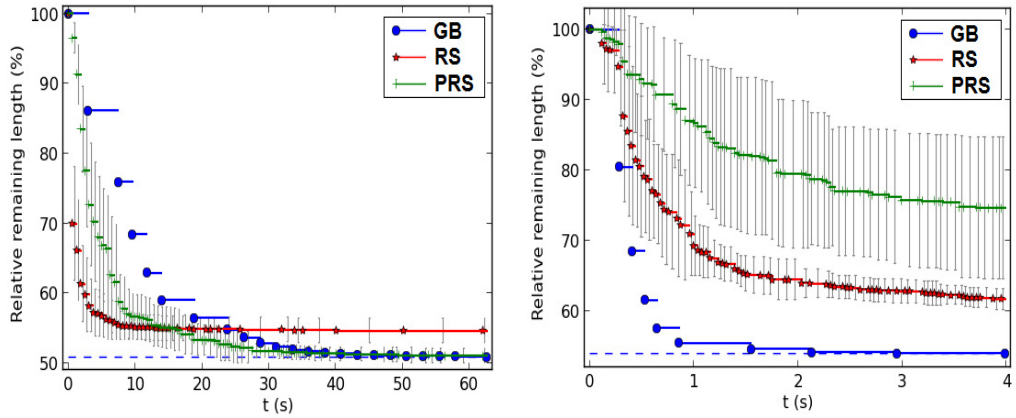
(d) Baxter-in-office

Figure 3.10: Influence of α_{init} on the convergence graphs of the GB optimizer. For each benchmark, the considered initial paths correspond to the ones of Figures 3.12 and 3.13.

3.7.2.3 Influence of a pruning preliminary step

For the UR5-industrial benchmark, we compared the optimization convergence graphs with and without a pruning step. Pruning was implemented following [Geraerts 2007], to remove redundant nodes in the initial path by creating valid shortcuts between the waypoints. RRT has been chosen as motion planner because it usually produces more waypoints than PRM, so the impact of pruning is more accountable. Results are given in Figure 3.11. The path length reduction of the three optimizers are still compared, but the notable information is the computation time of GB that is 16 times higher without pruning. Such lower computation time prevents the random optimizers to converge, so the GB result appears as better.

Note that, if pruning always reduces the GB optimization time, it sometimes spoils the path length reduction. In fact, there can be cases where multiple waypoints are useful to bypass an obstacle, rather than a long straight line in the configuration space.



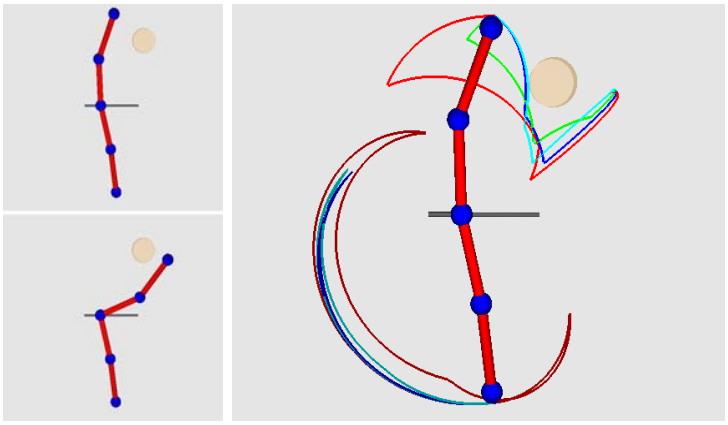
(a) Without pruning (b) With pruning

Figure 3.11: Influence of a pruning step on the optimization processes. RRT-connect provides an initial path with 62 waypoints, which is downed to 2 waypoints by Prune. Concerning the path length, it is only reduced of 6.2% by Prune. Thus, final path lengths provided by GB in both cases are equivalent, the major difference results in the computation time of GB.

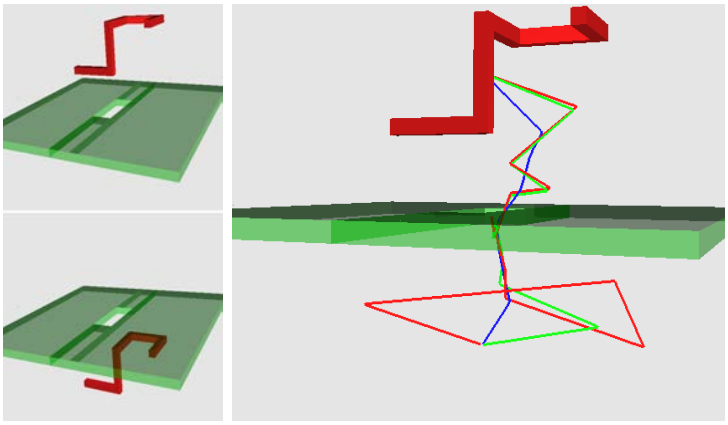
3.8 Conclusions

We managed to settle a path optimization for navigation and manipulation problems, and tested it with various robots and environments. Our algorithm uses standard numerical tools as collision checking, linearized one-dimensional constraint and LCQP resolution. It correlates them in a simple but effective way, and the algorithm structure is organized so that its convergence is guaranteed. Furthermore, our method only requires collision checking, therefore neither geometry pre-processing nor offline optimization are necessary to counterbalance costly distance computations. We demonstrate that the optimizer may be time-competitive compared to random shortcut in complex models where collision tests are time-consuming. It also proposes better quality paths, reducing the path length and removing unnecessary DOF motions. Finally, our optimizer manages to reduce a local detour in a long path while random shortcut methods will mostly fail.

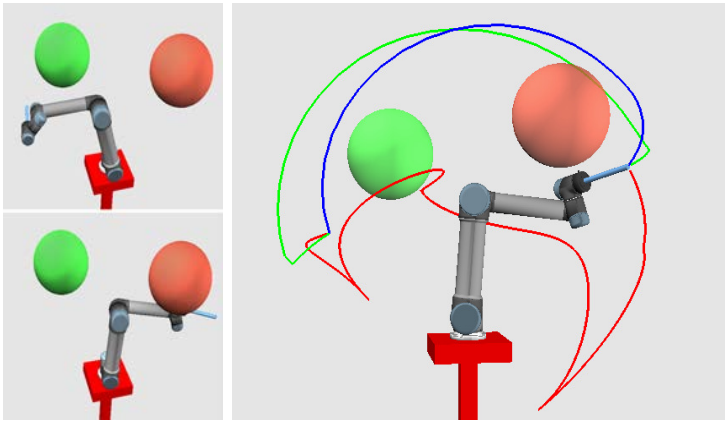
For future work, we have room for improvement. We can take advantage of the sparsity of the constraint Jacobian to reduce computation time. We may also adapt the iteration scalar parameter from geometrical considerations on the current path, e.g. using a lower bound of the distance between certain objects.



(a) Double-arms

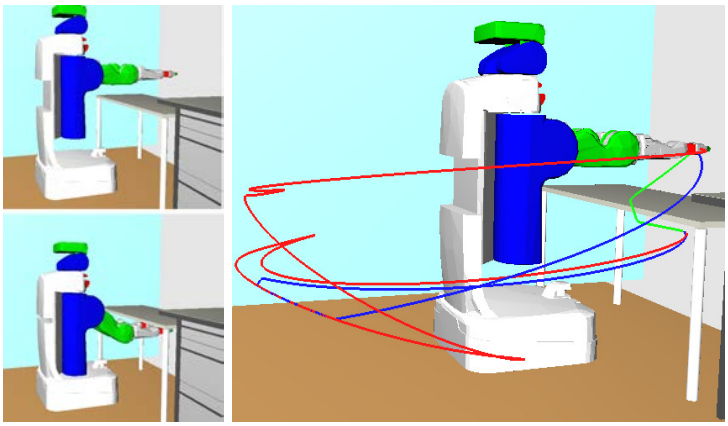


(b) Freeflyer-puzzle

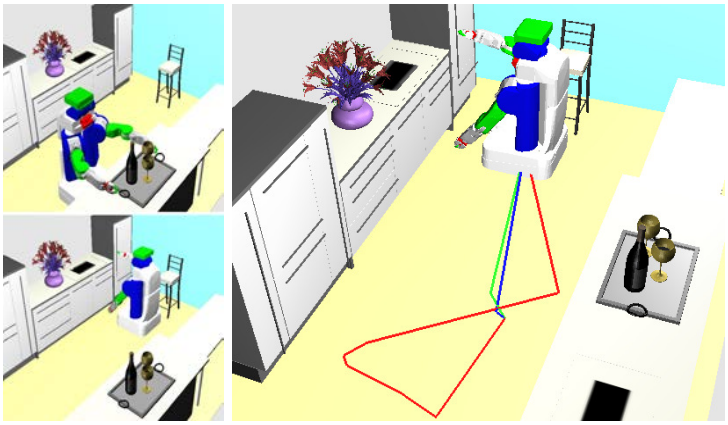


(c) UR5-with-spheres

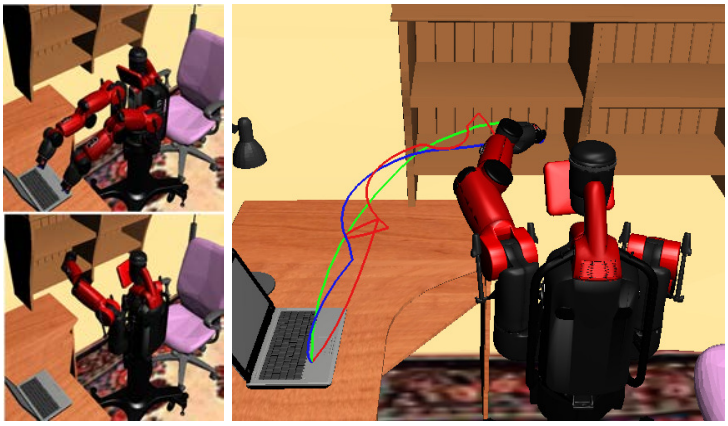
Figure 3.12: (Left) Initial and final configurations. (Right) Trajectories of end-effectors or centers along the different paths: the initial path is represented in red, the RS output in blue, the PRS output in cyan (top only) and the GB optimized path in green. The full robot motions can also be visualized in the joined video. The trajectory comparison highlights the optimization success of our method, which manages to deliver a shorter or similar path compared to the RS output. Note that, in the double-arms example, GB optimization also cancels the lower arm activation contrary to RS and PRS.



(a) PR2-in-kitchen-1

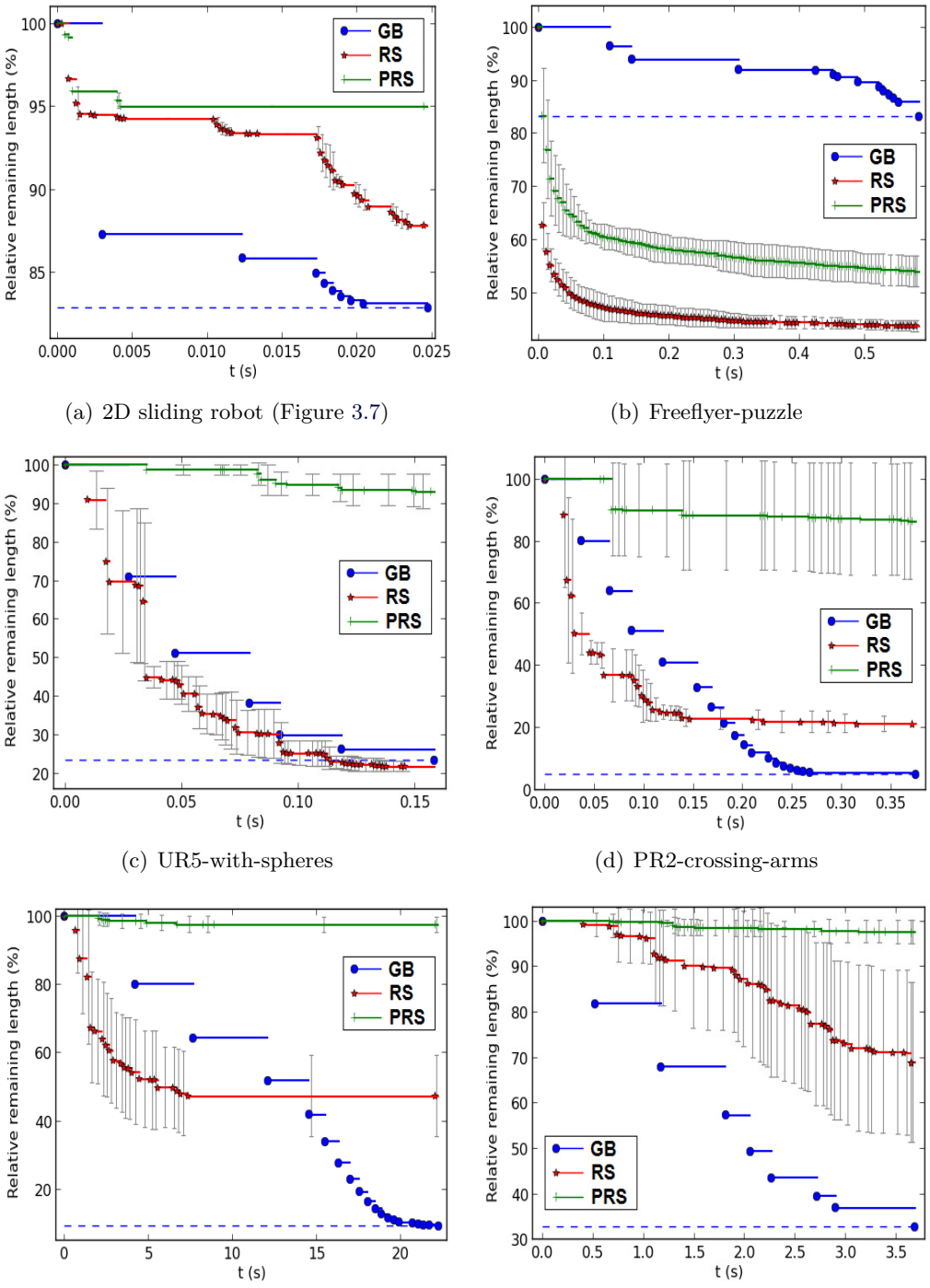


(b) PR2-in-kitchen-2



(c) Baxter-in-office

Figure 3.13: Other trajectories comparisons of end-effectors or mobile bases (initial path in red, RS output in blue, PRS output in cyan and GB output in green).



(a) 2D sliding robot (Figure 3.7)

(b) Freeflyer-puzzle

(c) UR5-with-spheres

(d) PR2-crossing-arms

(e) PR2-in-kitchen-1

(f) Baxter-in-office

Figure 3.14: Convergence graphs of the three optimizers during the optimization process. For each benchmark, the considered initial paths correspond to the ones of Figures 3.12 and 3.13. The remaining path length relative to the initial one is represented. The dashed blue line is the final result of GB. RS and PRS averages and standard deviations (in grey) are plotted for 50 launches.

Chapter 4

Jumping in robotics and computer animation

Contents

Ballistic motion synthesis has always been in roboticists' and animators' sights. Jumping increases the range of robot or character reachable space, and offers independence to the environment discontinuities. Furthermore, synthesizing highly dynamic motions is very challenging as these motions lie in the limits of the human capabilities [Edwardes 2009]. Figure 4.1 illustrates a jump from a parkour performance.

Figure 4.1: Execution of a parkour-style jump between to small walls. © Pixabay

While in robotics, robots have to be strong and agile enough to propel themselves in the air, in computer animation focus is done on the realism of the animations. This related work focuses on the different approaches to generate jump trajectories, borrowed to the two domains as they tackle the problem differently.

4.1 Jumping robots

In robotics, ballistic motion planning has been relatively little addressed, or in a simplified way. One-legged robots are hopping while keeping balance [Raibert 1984, Batts 2017], another miniature robot jumps to climb horizontal stairs [Stoeter 2005]. Similarly, the Sand Flea robot [Boston Dynamics 2012] is able to jump at a height of 10 meters using a CO_2 powered piston. A 2D multi-articulated gymnast robot jumps over obstacles on a horizontal surface, while taking into account the whole-body angular momentum [Papadopoulos 2007]. These works rely on horizontal surfaces and vertical obstacle clearance, which is constraining regarding the environment composition.

More recently, researchers have tried to design robots capable to execute parkour-like motions such as wall climbing in 2D [Degani 2014, Haldane 2016]. Focus is made on robot capacities to propel themselves in the air, and on re-orientation (see Figure 4.2). However, friction coefficients are artificially increased to reduce the risk of slippage, gravity effect may be lowered and contact planning is absent, the control relies on a state machine. The Handle robot [Boston Dynamics 2017] is rolling and jumping with its legs above obstacles or on flat surfaces, no motion planning seems to be involved for now.

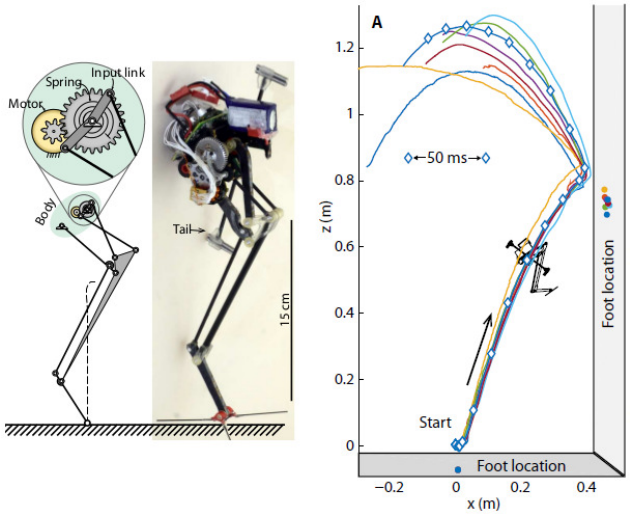


Figure 4.2: Robot presented by [Haldane 2016], and trajectories obtained when jumping from the ground to a vertical wall.

4.2 Motion planning techniques and data driven animation

Path planning is accurate when searching for a valid sequence of motions, including jumping, to reach a desired position. Synthesizing high quality motions for characters in such an application is challenging: the dimension of the problem equals

the number of DOFs, the environment is large and complex, and the motion is additionally constrained by the dynamics of the world. For this reason, motion synthesis is typically addressed with a decoupled approach. First, during the path planning phase, a collision-free path is found for the character using a sample-based planner. Then, during the animation phase, a motion is computed and played along the path.

A standard motion synthesis technique is to use pre-existing animations, produced either by 3D artists or from motion capture. These approaches are favored because of the high quality of the resulting animations [Kovar 2002] which are considered as *plausible* by the user.

The coordination of both planning and animation phases is critical to obtain plausible animations. This decoupling works well with stereotypical motions such as walking [Choi 2003, Pettré 2003, Esteves 2006, van Basten 2011]. Thanks to simplifying assumptions (periodic animations, contacts occurring with the ground...), the planned path for the character center easily extends to a trajectory at the animation phase, without considering the full dynamics of the model [Kajita 2003]. Contact generation or motion adaptation to changes of the environment or the character is then typically handled in the animation phase using local deformation methods on reference motions [Witkin 1995, Kovar 2002, Holden 2016]. However, they produce unnatural results when the deformation becomes too important.

It is also possible to analyze features that characterize the contact-rich motion repertoire of a character and to detect valid transitions in the environment where each of these motions may be possible and which surfaces will be used for support [Kapadia 2016]. This method is still limited to environments that fit with the motion database.

Animated motion planning has also been addressed with jumps [Yamane 2010]. This latter method computes a sequence of jumps whose heights are tuned in order to reach different levels while avoiding obstacles (see Figure 4.3).

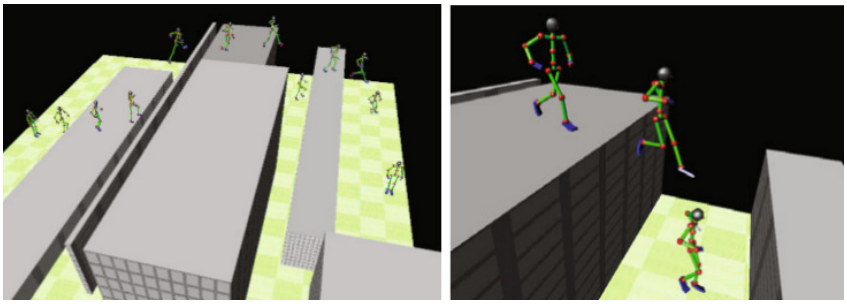


Figure 4.3: Snapshots of a planned motion including jumps, from the work of [Yamane 2010].

4.3 Physics-based motion synthesis

Rather than using reference animations, physics-based methods synthesize motions with algorithms based on a simplified model of the law of physics, which is solved most of the time with numerical optimization.

Space-time constraints is an old family of the physics-based techniques, claiming to replace part of manually defined animations so the motions look *real* at a basic mechanical level [Witkin 1988]. Space-time constraints can also be applied to create transitions in motion graphs, between segments of captured motions [Rose 1996, Arikan 2003, Safonova 2007].

Work by [Mordatch 2012] is focusing on automatic generation of contacts at the crossroad of motion synthesis and path planning. As the starting point and the modelization are crucial for the optimization to quickly converge toward plausible motions, motion capture often appears as a good initial guess [Safonova 2004, Levine 2011, Liu 2012]. In a similar way, existing motions may be edited of re-timed to consider physical objectives such as friction [Lamouret 1996, Pollard 2000, McCann 2006]. Finally, motion database can be coupled with a heuristic-based graph-search planner and trajectory optimization [Dellin 2012].

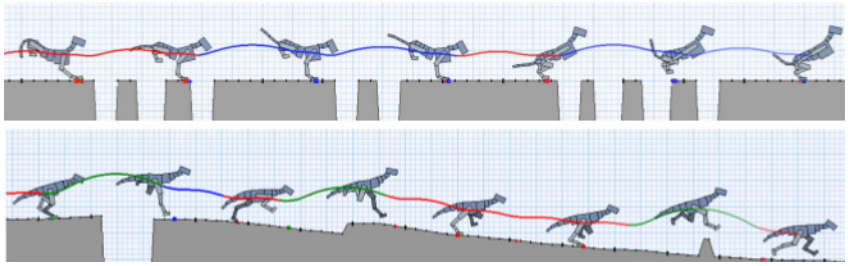


Figure 4.4: Trajectories of a jumping dog and a jumping raptor from [Peng 2016].

It is common for physics-based method to rely on proportional derivative (PD) controllers to compute the desired joint torques, once a target posture has been defined by the physics model and given a finite state machine. PD controllers have been used for athletic ballistic motions, for biped periodic locomotion and for dog jumps [Hodgins 1995, Wooten 1996, Yin 2007, Coros 2010, Coros 2011]. However PD controllers require a fine tuning of the parameters and, without a force model, simulated characters can exceed human capabilities. Plus, the number of states increases with the complexity of the environment and the desired motions, which can be bypassed with learning techniques. For instance, [Peng 2016, Liu 2016] synthesize online near optimal running and jumping motions for quadrupeds with reinforcement-based learning techniques (see Figure 4.4). Even so, they are based on simplifying assumptions regarding the location and periodicity of the contacts, which do not hold in arbitrary environments.

To guide manual design of jumping motions, physics-inspired methods display indications along the animation [Shapiro 2011]. The center of mass trajectory is shown as well as physically possible ones, regarding the execution time or the shape-

closeness to the initial trajectory (see Figure 4.5). This work also provides a tool to correct the global angular momentum from the limb motions to make the global orientation more realistic.

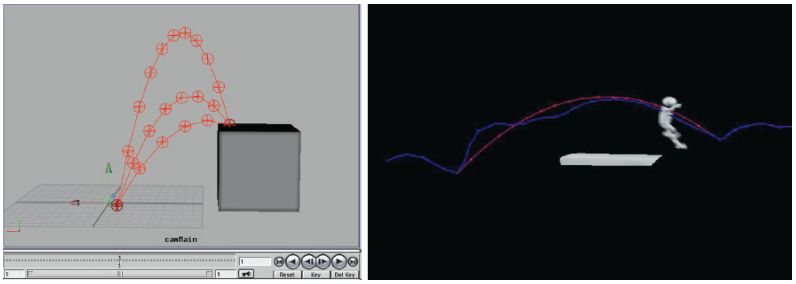


Figure 4.5: Illustrations of the ballistic shaping tool from [Shapiro 2011]. (Left) Example of the generation of multiple ballistic paths between two locators. (Right) One of these paths (in red) can serve to match a center of mass path (in blue) of a manually created trajectory.

Similarly optimization-based methods are not complete, and can get trapped in local optima [Mordatch 2012]. On top of data driven or physics-based animations, motion planning methods are required to provide the guarantee that a solution will be found in complex environments. Regarding motion planning, considering contact dynamics is not possible in the decoupled approach because it requires planning the contact locations simultaneously with the path. This introduces a combinatorial explosion of the computation time [Bretl 2004, Escande 2008]. The issue of generating relevant contacts along the motion is thus central and needs to be addressed properly.

Overall, some contributions that have studied jumping motions do not focus explicitly on path planning, rather on the physically accurate adaptation or synthesis of the jump animation. [Wensing 2014] introduce a simulated humanoid robot that runs and jumps on a horizontal platform. The takeoff leg angle and intensity are computed to cross the large gap. Similarly computer graphics contributions assume that the ballistic jumping motion is already precomputed [Reitsma 2003], and focus on the preparation phase [Sulejmanpašić 2005, Reitsma 2008], or the reception phase [Ha 2012]. New possibilities have also been explored to synthesize motions. For instance, exploiting the natural vibration modes of the body is able to produce walking and jump motions, without animating individual joints [Kry 2009].

To plan highly dynamic motions, recent contributions have proposed hybrid approaches, using both data-driven and physics-based methods to generate motions [Levine 2012, Tonneau 2016a]. In particular, they deform motion capture trajectories using physics-based heuristics, constraining the motion adaptation to respect the Euler equation of motion, given customizable bounds on the angular momentum of the character [Yamane 2010]. Because the contact locations are predefined relatively to the center of mass by the reference motion capture animation, the linear part is necessarily validated. However, this limitation once again prevents

generalization to arbitrary environments, where the contacts must be changed to obtain a valid motion.

4.4 Related work analysis

On one hand, despite promising results based on learning methods, data driven approaches do not generalize well to arbitrary environments. Because they rely on a limited set of reference motions, they are not complete (i.e. guaranteed to find a solution if it exists).

On the other hand, integrating dynamic models within sampling-based motion planners appears to be a difficult but necessary step to solve the motion synthesis problem in complex environments. Formerly, it has been necessary to integrate the dynamic properties of legged locomotion in motion planners, to ensure that the computed trajectories can be executed in a plausible manner. Our framework proposes a significant step in this direction, by extending a ballistic motion planner with the integration of a multi-contact dynamic model.

Additionally, it appears that neither data driven nor physics-based animation techniques are able to correctly compute contact locations when simplifying assumptions do not apply anymore (coplanar horizontal contacts, or predefined contact locations). Our framework is able to compute arbitrary contact configurations for such scenarios, based on our relaxed contact model.

Chapter 5

Ballistic motion planning for a point-mass

Contents

5.1 Problem statement

We consider the ballistic motion planning for a jumping robot in 3D environments containing slippery surfaces. It is well-known that ballistic motion results in a parabola trajectory. According to the Coulomb friction law, a condition for the robot not to slide during its takeoff is that the contact force belongs to a so-called friction cone. This latter property extends to the landing phase. We consider a point-mass robot with simplified contact dynamics: we assume that the robot is submitted to an impulse force as soon as it lands, so that the transition between landing and takeoff is instantaneous. This gives rise to a discontinuity between the contact forces and the contact velocities. Moreover we assume that the robot has limited energy resources, which limit the velocity at takeoff. The landing velocity is also constrained to avoid requiring to dissipate too much energy (and damaging the robot). These energy restrictions are realized by limiting the magnitude of the

velocity vectors during the takeoff and landing phases. Constraints on the velocity vector magnitudes are named velocity constraints.

The contribution of this chapter is to design an algorithm, the Ballistic Motion Planner, which is able to plan a collision-free path satisfying both sliding and velocity constraints in such a context. This chapter does not consider the full dynamics of articulated avatars, but is restricted to point-robots. With respect to the state of the art, the contribution is to account for slipping prevention as well as takeoff and landing velocity limitations. Furthermore, the proposed approach applies on 3D environments and rough terrains without any approximation, prior knowledge or restriction.

Let us consider a point-robot moving in a 3D environment. The robot begins from a starting position $\mathbf{c}_s$ and wants to reach a goal position $\mathbf{c}_g$, only by performing jumps from one contact to another. Both $\mathbf{c}_s$ and $\mathbf{c}_g$ are assumed to be in contact with the environment. There is no distinction between ground and obstacles. The purpose of this method is to determine a sequence of jumps, under the following assumptions:

- The robot is modeled by a point mass m of position $\mathbf{c}$ with respect to the origin.
- The only force that applies to the robot during a jump is $m\mathbf{g}$ where $\mathbf{g} = (0\ 0\ -g)^T$, $g = -9.81$. No air drag is considered.
- Contact phases are instantaneous, so that the velocity at a contact point is discontinuous, i.e. transition from landing to takeoff results from an impulsion.
- The surface material is uniform in the environment, i.e. the non-sliding constraints can be modeled everywhere by a friction cone with a constant coefficient. We denote by μ the tangent of the cone half-angle.
- Takeoff and landing velocity magnitudes are bounded by the same value, so that an admissible jump path can be followed in a reversed direction.
- There is no constraint on the energy balance between one jump and the next.
- The robot cannot collide with the obstacles.

Figure 5.1 illustrates the effects of the friction and velocity constraints on the existence of parabola sequences. The following section reminds the basics of ballistic motion and details the equations of parabolas linking two points.

5.2 Unconstrained ballistic motion

5.2.1 Accessible space of ballistic motion

We denote the global frame basis by $(\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z)$. When Newton's second law of motion is integrated with respect to time for a ballistic shot from the $\mathbf{c}_s$ position

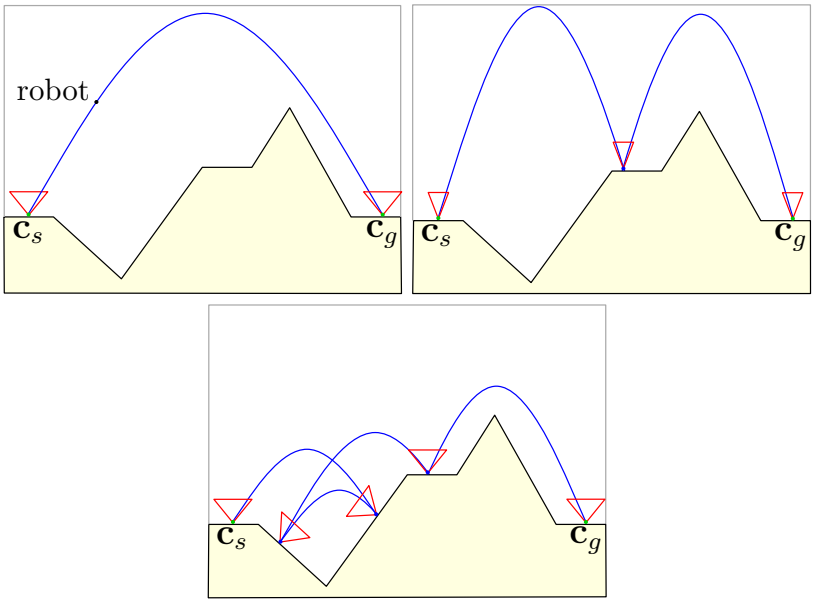


Figure 5.1: Three sequences of parabolas between $\mathbf{c}_s$ and $\mathbf{c}_g$ positions for different constraints. In the case illustrated on the right, friction cones are narrower than those on the left, so that the parabola on top is not admissible anymore, and a waypoint has to be used. In the bottommost case, the initial velocity has been limited compared to the left case, resulting in a sequence with numerous jumps.

with a $\dot{\mathbf{c}}_s$ initial velocity, the following robot trajectory is obtained:

$$\mathbf{c}(t) = -\frac{g}{2}t^2\,\mathbf{e}_z + \dot{\mathbf{c}}_s\,t + \mathbf{c}_s \tag{5.1}$$

Let $(x\ y\ z)^T$ be the coordinates of $\mathbf{c}$. The ballistic motion belongs to a vertical plane denoted by π_θ. The orientation of the plane is given by the initial velocity components as follows:

$$\theta = \text{atan2}(\dot{y}_s, \dot{x}_s) \in [-\pi, \pi]$$

Considering $\mathbf{\Theta} = (\cos(\theta)\ \ \sin(\theta)\ \ 0)^T$, we introduce the following variable changes involving the scalar product:

$$x_\theta = \mathbf{c} \cdot \mathbf{\Theta}, \quad x_{\theta_s} = \mathbf{c}_s \cdot \mathbf{\Theta}$$

$$x_{\theta_g} = \mathbf{c}_g \cdot \mathbf{\Theta}, \quad \dot{x}_{\theta_s} = \dot{\mathbf{c}}_s \cdot \mathbf{\Theta}$$

Thus from Equation (5.1), one can rewrite the main equations of motion determining the robot coordinates $(x_\theta\ z)^T$ in π_θ (see Figure 5.2):

$$z = -\frac{g}{2}\frac{(x_\theta - x_{\theta_s})^2}{\dot{x}_{\theta_s}^2} + \frac{\dot{z}_s}{\dot{x}_{\theta_s}}(x_\theta - x_{\theta_s}) + z_s \tag{5.2}$$

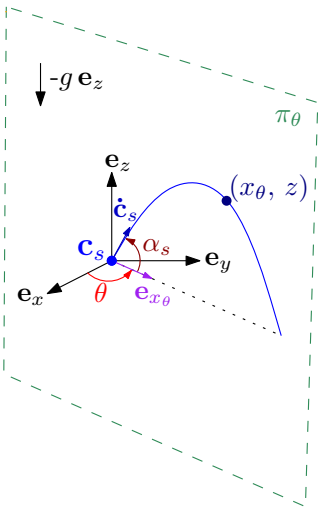


Figure 5.2: The parabola always belongs to the plane π_θ defined by $(\mathbf{c}_s, \mathbf{e}_{x_\theta}, \mathbf{e}_z)$, where $\mathbf{e}_{x_\theta} = \cos(\theta)\mathbf{e}_x + \sin(\theta)\mathbf{e}_y$.

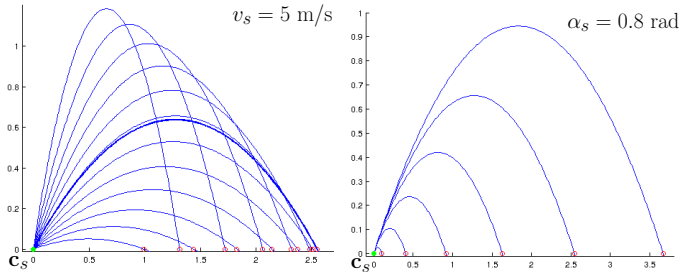


Figure 5.3: Accessible space from $\mathbf{c}_s = (0\ 0)^T$ when varying takeoff angle α_s (left), or when varying initial velocity v_s (right). On left, the bold parabola leads to the maximal range at given initial velocity, and is obtained for $\alpha_s = \frac{\pi}{4}$.

$$\frac{\dot{z}}{\dot{x}_{\theta_s}} = -g\frac{x_\theta - x_{\theta_s}}{\dot{x}_{\theta_s}^2} + \frac{\dot{z}_s}{\dot{x}_{\theta_s}} \tag{5.3}$$

Let us denote the takeoff angle by $\alpha_s = \text{atan2}(\dot{z}_s, \dot{x}_{\theta_s})$ and the velocity value $||\dot{\mathbf{c}}_s||$ by v_s. Equations (5.2-5.3) highlight the two parameters α_s and $\dot{x}_{\theta_s}$ that determine a parabola in π_θ. For instance, Figure 5.3 presents the parabola beams when v_s (resp. α_s) is fixed. This can also be viewed as the accessible space of the robot performing ballistic motions.

5.2.2 Goal-oriented ballistic motion

Now we want our robot to reach the goal position $\mathbf{c}_g$ with a jump starting from $\mathbf{c}_s$. Therefore, the value $\theta = \text{atan2}(y_g - y_s, x_g - x_s)$ is now known. Let X_θ equal $x_{\theta_g} - x_{\theta_s}$ and Z equal $z_g - z_s$. Since Z is fixed, it appears that from Equation (5.2), α_s is the only remaining variable to compute the parabola beam leading to $\mathbf{c}_g$. In

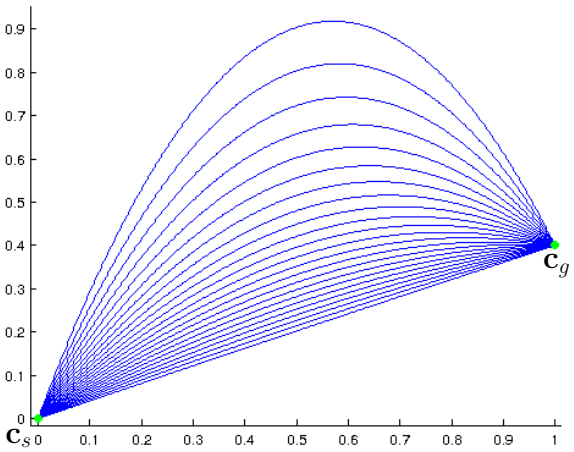


Figure 5.4: Physically-feasible parabolas linking $\mathbf{c}_s$ and $\mathbf{c}_g$, for multiple values of α_s in $[0.91, 1.27]$ rad.

fact, the initial velocity $\dot{x}_{\theta_s}$ can be obtained with the following equation:

$$\dot{x}_{\theta_s} = \sqrt{\frac{gX_\theta^2}{2(X_\theta \tan(\alpha_s) - Z)}} \tag{5.4}$$

Equation (5.4) implies that $\dot{x}_{\theta_s}$ is only defined for top-curved parabolas, which is consistent with gravity. Non-physically-feasible parabolas such as down-curved ones are not considered. Therefore we impose:

$$\text{atan2}(Z, X_\theta) < \alpha_s < \frac{\pi}{2} \tag{5.5}$$

An example of a goal-oriented parabola beam is presented Figure 5.4. Finally, we denote a parabola starting from $\mathbf{c}_s$ and its parameters by $\mathcal{P}_s(\theta, \alpha, v)$.

5.3 Ballistic motion with constraints

So far we have defined a beam of feasible parabolas to connect two positions. In this section, the non-sliding and velocity constraints are introduced, and the resulting reduction of the space of admissible parabola beams is detailed.

5.3.1 Non-sliding constraints

5.3.1.1 Impulse model

Let us consider one point $\mathbf{c}$ at the contact of an environment surface. The surface normal is denoted by $\mathbf{n}$. We may takeoff from this point, landing to this point or make a transition between two jumps at this point. We denote by $\dot{\mathbf{c}}_t$ and $\dot{\mathbf{c}}_l$ respectively the takeoff and landing velocity vectors. The instantaneous velocity

shift $d\dot{\mathbf{c}}$ is constrained by the relationship:

$$d\dot{\mathbf{c}} = \dot{\mathbf{c}}_t - \dot{\mathbf{c}}_l \tag{5.6}$$

Under the conservative non-slipping condition, Newton's equation is written:

$$m\frac{d\dot{\mathbf{c}}}{dt} - m\mathbf{g} = \mathbf{f_c}$$

with $\mathbf{f_c}$ the impulse contact force applied by the point-robot on the surface. Under the impulse hypothesis dt is instantaneous, such that the action of gravity is not measurable:

$$m\, d\dot{\mathbf{c}} \approx \mathbf{f_c} dt$$

Thus, $d\dot{\mathbf{c}}$ and $\mathbf{f_c}$ are colinear. From here, takeoff and landing have to be executed without slippage. According to the friction law, this implies that the contact force $\mathbf{f_c}$ has to belong to the friction cone $\mathcal{K}_\mathbf{c}$ of the surface containing $\mathbf{c}$:

$$||\mathbf{f_c} - (\mathbf{f_c} \cdot \mathbf{n})\,\mathbf{n}|| \geq \mu\,(\mathbf{f_c} \cdot \mathbf{n})$$

We make the following observation that, by definition of a convex cone:

$$-\dot{\mathbf{c}}_l \in \mathcal{K}_\mathbf{c} \text{ and } \dot{\mathbf{c}}_t \in \mathcal{K}_\mathbf{c} \implies d\dot{\mathbf{c}} \in \mathcal{K}_\mathbf{c} \tag{5.7}$$

Proposition (5.7) provides a sufficient condition for satisfying the non-slipping condition: the velocity vectors have to lie in the friction cone so that the non-sliding constraint is respected. The benefit of this conservative condition will be detailed in the motion planner section.

A non-slipping condition benefit is presented in Figure 5.5, where the absence of friction constraints results in an unnatural trajectory. We use the non-slipping condition for validating jumping trajectories in the motion planner we present in a further section.

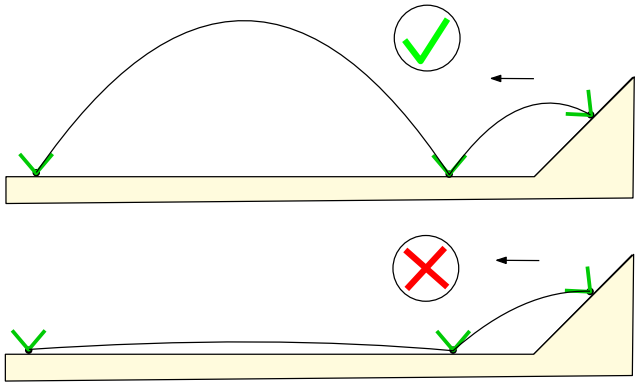


Figure 5.5: Illustration of the conservative non-slipping condition. (Top) The trajectory is valid, since landing and takeoff velocties are included in the centroidal green cone. (Bottom) The trajectory is found invalid by our criterion.

5.3.1.2 2D reduction

Non-sliding constraints impose the robot to land and take off along velocity vectors that belong to 3D friction cones of apexes based on $\mathbf{c}_s$ and $\mathbf{c}_g$.

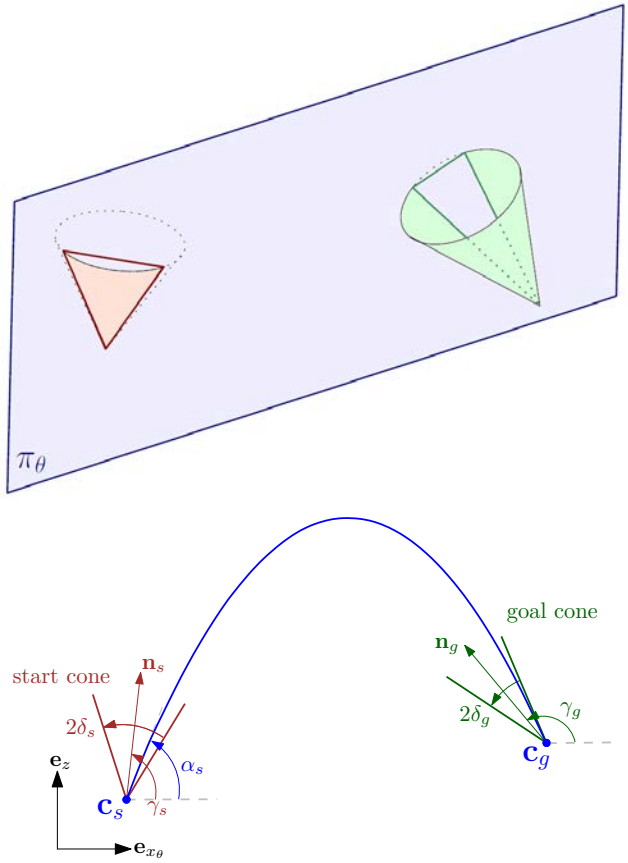


Figure 5.6: (Top) Representation of the intersection between π_θ and two 3D cones. (Bottom) 2D cones resulting of the intersection and an example of parabola belonging to both cones.

Since the motion has to lie in a vertical plane π_θ, the problem of computing a parabola between the two 3D friction cones is reduced to a 2D problem. Corresponding 2D cones result from the intersections of the 3D cones with the plane π_θ (see Figure 5.6 top). If one of both intersection sets is reduced to a point, there is no possible jump between $\mathbf{c}_s$ and $\mathbf{c}_g$. Otherwise, let us denote their half-apex angles respectively by δ_s and δ_g, and their directions projected in π_θ respectively by $\mathbf{n}_s$ and $\mathbf{n}_g$. Detailed computation of δ_s and δ_g is presented in Equation (A.5) of Appendix A. Note that δ_s and δ_g may be smaller that $\arctan(\mu)$. For the 2D start cone of direction $\mathbf{n}_s = (n_{x_s}\ n_{y_s}\ n_{z_s})^T$, let us denote by γ_s the angle between the cone direction and the horizontal line:

$$\gamma_s = \text{atan2}(n_{z_s}, n_{x_s}\cos(\theta) + n_{y_s}\sin(\theta))$$

γ_g is similarly defined, respectively to the 2D goal cone. Thus the problem of slippage avoidance is reduced to the problem of finding a parabola going through the 2D cones (see Figure 5.6 bottom).

5.3.2 Constraint formulation

Since the equation of a parabola starting at $\mathbf{c}_s$ and ending at $\mathbf{c}_g$ only depends on α_s, the four constraints are just formulated relatively to α_s.

For the non-sliding takeoff constraint, the inequalities on α_s are immediate:

$$\alpha_1^- \leq \alpha_s \leq \alpha_1^+ \quad \text{with} \begin{cases} \alpha_1^- = \gamma_s - \delta_s \\ \alpha_1^+ = \gamma_s + \delta_s \end{cases}$$

To express the three remaining constraints according to α_s, Equations (5.2-5.3) are brought back to the parabola origin $\mathbf{c}_s$. Thus constraints are still expressed as inequalities:

$$\alpha_i^- \leq \alpha_s \leq \alpha_i^+ \ , \ i \in \{2..4\}$$

For the landing cone constraint, different cases appear, depending on the accessibility of the cone. They are tackled by Algorithm 4, which returns the constraint bounds (α_2^-, α_2^+). Note that one of the bounds may not exist, and that the constraint may also not be satisfied.

5.3.3 Velocity constraints

Takeoff velocity limitation is expressed as $v_s \leq V_{max}$. V_{max} is manually chosen according to the maximal allowed jump range X^{max} (e.g. on a flat ground, $X^{max} = V_{max}^2/g$). Equation (5.2) leads to:

$$gX_\theta^2 \tan(\alpha_s)^2 - 2X_\theta V_{max}^2 \tan(\alpha_s) + gX_\theta^2 + 2ZV_{max}^2 \leq 0 \tag{5.8}$$

Let us set:

$$\Delta = V_{max}^4 - 2gZV_{max}^2 - g^2X_\theta^2$$

If $\Delta < 0$, Equation (5.8) has no solution. In other words, it means that the goal position is not reachable with an initial velocity satisfying the limitation. In the case where the constraint is solvable, we write:

$$\begin{cases} \alpha_3^- = (V_{max}^2 - \sqrt{\Delta})/gX_\theta \\ \alpha_3^+ = (V_{max}^2 + \sqrt{\Delta})/gX_\theta \end{cases}$$

The same argument can be applied for the landing velocity limitation (see Figure 5.7). Using Equations (5.3-5.4), we can rewrite the constraint equation $v_f \leq V_{max}$ as:

$$gX_\theta^2 \tan(\alpha_s)^2 - (4X_\theta Zg + 2X_\theta V_{max}^2)\tan(\alpha_s) + gX_\theta^2 + 2ZV_{max}^2 + 4gZ^2 \leq 0 \tag{5.9}$$

Algorithm 4 Resolution of the landing cone constraint.

Output: Defined constraint bounds α_2^-, α_2^+
  if $\gamma_g > 0$ **then**
    $\alpha_g^- = \gamma_g - \pi - \delta_g$
    $\alpha_g^+ = \gamma_g - \pi + \delta_g$
    if $\alpha_g^+ < -\frac{\pi}{2}$ **then**
      No solution
    else
      $\alpha_2^- = \arctan(\frac{2Z}{X_\theta} - \tan(\alpha_g^+))$
      if $\alpha_g^- > -\frac{\pi}{2}$ **then**
        $\alpha_2^+ = \arctan(\frac{2Z}{X_\theta} - \tan(\alpha_g^-))$
      else
        α_2^+ not defined
  else
    $\alpha_g^- = \gamma_g + \pi - \delta_g$
    $\alpha_g^+ = \gamma_g + \pi + \delta_g$
    if $\alpha_g^+ > \frac{\pi}{2}$ **then**
      No solution
    else
      $\alpha_2^+ = \arctan(\frac{2Z}{X_\theta} - \tan(\alpha_g^-))$
      if $\alpha_g^+ < \frac{\pi}{2}$ **then**
        $\alpha_2^- = \arctan(\frac{2Z}{X_\theta} - \tan(\alpha_g^+))$
      else
        α_2^- not defined

Let us set:

$$\Lambda = V_{max}^4 + 2gZV_{max}^2 - g^2X_\theta^2$$

If $\Lambda < 0$, Equation (5.9) has no solution. Otherwise, we write:

$$\begin{cases} \alpha_4^- = (V_{max}^2 + 2gZ - \sqrt{\Lambda})/gX_\theta \\ \alpha_4^+ = (V_{max}^2 + 2gZ + \sqrt{\Lambda})/gX_\theta \end{cases}$$

A symmetry property of the parabola implies that, given one parabola from $\mathbf{c}_s$ to $\mathbf{c}_g$ determined by $\dot{\mathbf{c}}_s$, the same parabola can be obtained from $\mathbf{c}_g$ to $\mathbf{c}_s$ with $-\dot{\mathbf{c}}_g$ as initial velocity. In the latter case, the contact velocity becomes $-\dot{\mathbf{c}}_s$. We use this property to apply the same bound V_{max} for the takeoff and landing velocities. Thus the parabola can be traveled both ways without violating the velocity limitation constraints.

5.3.4 Constraint collection and solution existence

The domains where constraints are satisfied are convex. Thus, we intersect these domains to determine if an interval $]\alpha_s^-, \alpha_s^+[$ of α_s values complying with all the

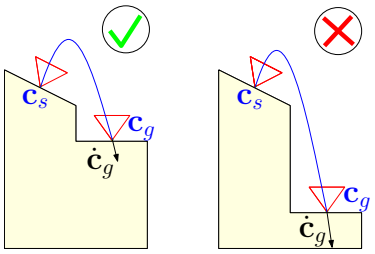


Figure 5.7: The landing velocity limitation allows to reject parabolas that have too important an impact velocity magnitude v_g (right).

constraints exists. The interval bounds are given by:

$$\begin{cases} \alpha_s^- = \max(\alpha_1^-, \alpha_2^-, \alpha_3^-, \alpha_4^-) \\ \alpha_s^+ = \min(\alpha_1^+, \alpha_2^+, \alpha_3^+, \alpha_4^+) \end{cases}$$

Note that Equation (5.5) has to be simultaneously satisfied to consider an admissible parabola. Figure 5.8 presents an illustration of this constraint intersection. Constraint bounds $(\alpha_i^-, \alpha_i^+)_{i\in\{1..4\}}$ are used to plot parabolas, representing the domains where constraints are satisfied. The intersection of these domains leads to the set of possible solutions.

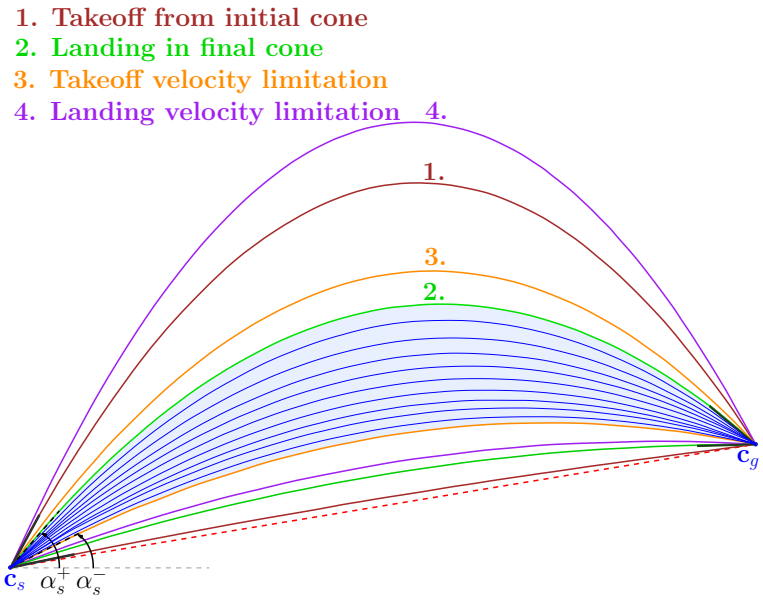


Figure 5.8: Illustration of the constraints on a practical example: each constraint bound is used as α_s and represents a bold parabola. Between these bounds, the constraint is satisfied, out of them not. The constraint intersection is given by the bounds (α_s^-, α_s^+) and illustrated by the gray zone: blue parabolas belonging to it are admissible solutions to the problem.

Finally, the existence of an admissible jump between two points is guaranteed as soon as:

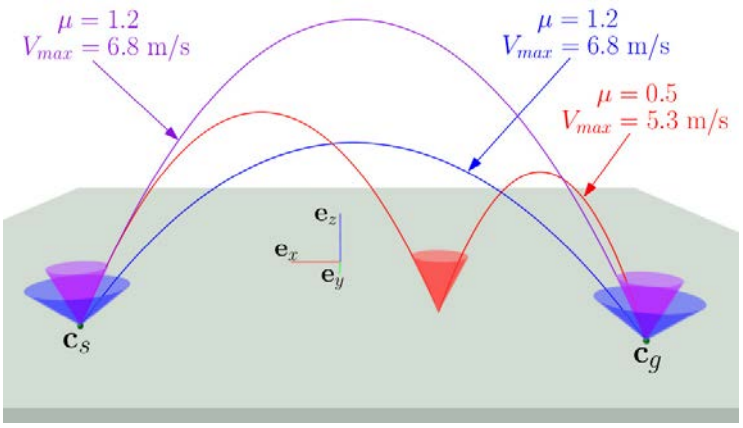


Figure 5.9: Three parabola examples linking $\mathbf{c}_s$ to $\mathbf{c}_g$ with different constraints. Large (blue) and narrow (violet) friction cones are considered, forcing the solution parabola to be adapted. With a large velocity limitation, $\mathbf{c}_g$ can be directly reached (blue). Otherwise, an intermediate position has to be considered (red).

- Neither of the intersections between both friction cone and π_θ is reduced to a point.
- (α_s^-, α_s^+) are defined and $\alpha_s^- \leq \alpha_s^+$.

The two conditions are necessary and sufficient. The interval $]\alpha_s^-, \alpha_s^+[$ gives a simple parametrization of the solution beam. Choosing α_s as the average $0.5(\alpha_s^- + \alpha_s^+)$ allows to optimize the distance to the constraints, e.g. to be far from the limits of the friction cones, and so far from sliding. Figure 5.9 illustrates the constraint effects on a simple example.

5.4 Motion planning algorithm

5.4.1 Algorithm

To find a sequence of parabola arcs between an initial position $\mathbf{c}_s$ and a final one $\mathbf{c}_g$, we use a dense PRM-based Probabilistic Roadmap Planner [Kavraki 1996] (see Algorithm 5), where the roadmap may contain cycles. The planner can be used offline to explore an unknown environment and build a roadmap in the 3D space by randomly sampling contact positions (RANDOMSAMPLE). The surface sampling is conducted similarly to [Amato 1996] to have a normalized repartition of the samples among the obstacle surfaces, modelized by triangles. Then, positions are linked (STEER) with admissible collision-free parabola arcs. The roadmap construction is over as soon as either a path linking $\mathbf{c}_s$ and $\mathbf{c}_g$ is found (ARECONNECTED), or computation time is over. Finally, the function FINDSHORTESTPATH explores the roadmap to return the shortest path sequence, in terms of sum of parabola lengths. As no symbolic expression exists to compute the parabola lengths, a Simpson quadrature

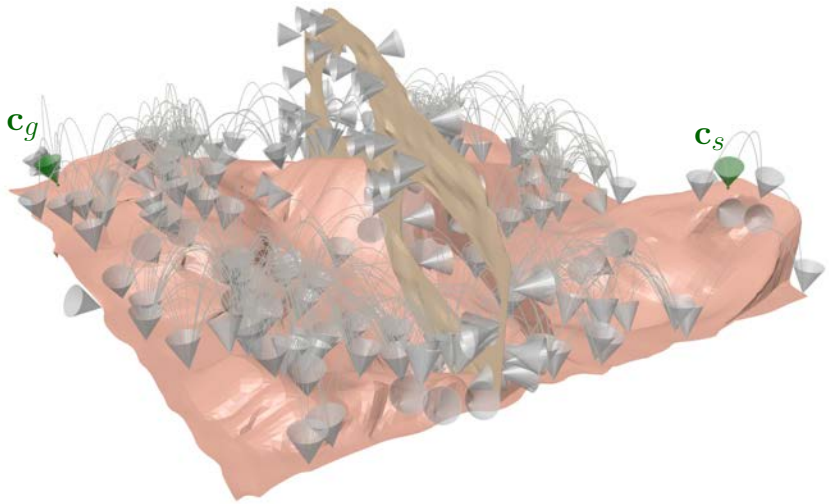


Figure 5.10: Preliminary result of a roadmap generation in a desert environment. Nodes are represented by their friction cones.

Algorithm 5 Probabilistic roadmap planner for ballistic motion planning.

Input: *environment*, $\mathbf{c}_s$, $\mathbf{c}_g$, μ, V_{max}
Output: Collision-free solution *sequence* to problem

```
path ←STEER(c_s, c_g)
finished ← ARECONNECTED(c_s, c_g)
while not(finished) do
    c_random ← RANDOMSAMPLE()
    ADDTOROADMAP(c_random)
    for c_node ∈ Roadmap do
        path ←STEER(c_node, c_random)
        ADDTOROADMAP(path)
    finished ← ARECONNECTED(c_s, c_g)
return sequence ← FINDSHORTESTPATH()
```

of order 6 is used to compute them numerically. An example of ballistic roadmap is given in Figure 5.10.

Note that the sufficient non-slipping condition reduces the dimensionality of the motion planning problem, because it removes the relationship between the entering velocity and the exiting velocity of a node. With a classic kynodynamic planner [Kunz 2014], to verify whether a trajectory can be connected with another one, it is required to extend the state space with the velocities and so it doubles the dimensionality of the problem. In our case, this is only required to verify whether there exists a velocity vector belonging to the cone each time we want to add a new path, independently of other paths.

The function BEAM is described in Algorithm 6. It computes the interval of takeoff angles that generate constrained parabolas to link $\mathbf{c}_s$ and $\mathbf{c}_g$. The algorithm

Algorithm 6 BEAM($\mathbf{c}_s, \mathbf{c}_g$): Computes the parabola beam represented by the takeoff angle interval $]\alpha_s^-, \alpha_s^+[$.

Input: $\mathbf{c}_s$, $\mathbf{c}_g$, μ, V_{max}
Output: Interval of takeoff angles I_{beam}
  $cone_s^{2D} \leftarrow$ COMPUTEINTERSECTION($cone_s^{3D}, \pi_\theta$)
  $cone_g^{2D} \leftarrow$ COMPUTEINTERSECTION($cone_g^{3D}, \pi_\theta$)
  if ISREDUCED($cone_s^{2D}$) **or** ISREDUCED($cone_g^{2D}$) **then**
    return $I_{beam} \leftarrow \emptyset$
  $(\alpha_i^- \alpha_i^+, fail)_{i\in\{1..4\}} \leftarrow$ COMPUTECONSTRAINTS()
  if $fail = true$ **then return** $I_{beam} \leftarrow \emptyset$
  $\alpha_s^- \leftarrow \max(\alpha_1^-, \alpha_2^-, \alpha_3^-, \alpha_4^-)$
  $\alpha_s^+ \leftarrow \min(\alpha_1^+, \alpha_2^+, \alpha_3^+, \alpha_4^+)$
  return $I_{beam} \leftarrow]\alpha_s^-, \alpha_s^+[$

starts by calculating each cone and plane π_θ intersection, and continues only if both intersections are not reduced to the cone apexes. Then, takeoff angle bounds related to constraints are computed as presented in Section 5.3. A boolean $fail$ conveys the feasibility of constraints, i.e. if one constraint cannot be satisfied, $fail$ is set to $true$. At this stage, an admissible parabola exists if the global constraint bounds verify $\alpha_s^- \leq \alpha_s^+$.

Then the steering method STEER detailed in Algorithm 7 selects a takeoff angle α_s and tests the corresponding parabola for collisions. If the parabola is not collision-free (HASCOLLISIONS), then we select a new parabola α_s by dichotomy on the interval $]\alpha_s^-, \alpha_s^+[$ until the resolution threshold n_{limit} is reached. In the worst case, the DICHOTOMY function allows to span the almost entire parabola beam. Doing so, the algorithm is probabilistically complete as proven by the following property.

5.4.2 Probabilistic convergence study

The two following properties prove the probabilistic convergence of our algorithm, under some given assumptions.

Property 3 (Topological property)**.** *Let us consider an admissible parabola $\mathcal{P}_s(\theta, \alpha, v)$ starting at $\mathbf{c}_s$ and ending at $\mathbf{c}_g$. There exists a neighborhood $\mathcal{N}_s$ (resp. $\mathcal{N}_g$) of $\mathbf{c}_s$ (resp. $\mathbf{c}_g$) such that any pair of points ($\mathbf{c}_s^*$,$\mathbf{c}_g^*$) belonging to $\mathcal{N}_s \times \mathcal{N}_g$ can be linked by an admissible parabola.*

Proof. Let us consider the parabola family $\{\mathcal{P}_s(\theta + e_1, \alpha + e_2, v + e_3),\ e_i \in]-\varepsilon, \varepsilon[\}$ starting at $\mathbf{c}_s$. The function giving the three parabola parameters from the three coordinates of $\mathbf{c}_g$ is an homeomorphism. Therefore such a family spans a neighborhood of $\mathbf{c}_g$. Let $\mathbf{c}_g^*$ be a point of this neighborhood and $\mathcal{P}_s(\theta + e_1^*, \alpha + e_2^*, v + e_3^*)$ the parabola from $\mathbf{c}_s$ to $\mathbf{c}_g^*$. $\mathbf{c}_g^*$ may be chosen close enough from $\mathbf{c}_g$ to guarantee that e^* is small enough, and then $\mathcal{P}_s(\theta + e_1^*, \alpha + e_2^*, v + e_3^*)$ is admissible. Because the con-

Algorithm 7 STEER($\mathbf{c}_s, \mathbf{c}_g$): Steering method based on a constrained parabola. Returns a collision-free path linking $\mathbf{c}_s$ and $\mathbf{c}_g$. Otherwise, returns an empty path.

Input: $\mathbf{c}_s$, $\mathbf{c}_g$, μ, V_{max}, n_{limit}
Output: Collision-free parabola path *path*

```
]α_s^-, α_s^+[ ← I_beam
n ← 1
I_beam ← BEAM(c_s, c_g)
if ISEMPTY(I_beam) then return path ← emptyPath
else
α_s ← 0.5(α_s^- + α_s^+)
path ← COMPUTEPARABOLA(c_s, c_g, α_s)
while HASCOLLISIONS(path) and n < n_limit do
    α_s ← DICHOTOMY(]α_s^-, α_s^+[, n)
    path ← COMPUTEPARABOLA(c_s, c_g, α_s)
    n ← n + 1
if HASCOLLISIONS(path) then path ← emptyPath
return path
```

struction is symmetric, let us consider the same parabola as starting at $\mathbf{c}_g^*$ and ending at $\mathbf{c}_s$. We get a new parametrization of the same parabola, i.e. $\mathcal{P}_g(\theta^*, \alpha^*, v^*)$. By using the same argument as above, the parabola family $\{\mathcal{P}_g(\theta^* + e_1, \alpha^* + e_2, v^* + e_3)$, $e \in]-\varepsilon, \varepsilon[\}$ starting at $\mathbf{c}_g^*$ spans a neighborhood of $\mathbf{c}_s$. The property holds for any point $\mathbf{c}_g^*$ sufficiently close to $\mathbf{c}_g$. Therefore, there exists a neighborhood $\mathcal{N}_s$ (resp. $\mathcal{N}_g$) of $\mathbf{c}_s$ (resp. $\mathbf{c}_g$) such that any pair of points ($\mathbf{c}_s^*$,$\mathbf{c}_g^*$) belonging to $\mathcal{N}_s \times \mathcal{N}_g$ can be linked by an admissible parabola. □

Property 4 (Probabilistic convergence)**.** *Let us consider a sequence of collision-free ballistic jumps between two points $\mathbf{c}_s$ and $\mathbf{c}_g$ in a given environment. Let us assume that the entire path is at a distance of about ε from the obstacles. Then the probability for Algorithm 5 to find a sequence of collision-free ballistic jumps between $\mathbf{c}_s$ and $\mathbf{c}_g$ converges to 1 when running time tends to infinity.*

Proof. The property 4 is a direct consequence of the Property 3. Indeed, let us consider a sequence of collision-free ballistic jumps between two points $\mathbf{c}_s$ and $\mathbf{c}_g$. Let $\mathbf{c}_i$ and $\mathbf{c}_{i+1}$ two consecutive points in the sequence. $\mathbf{c}_i$ and $\mathbf{c}_{i+1}$ are linked by a collision-free parabola $\mathcal{P}_i$. From the topological property, there are two neighborhoods $\mathcal{N}_i$ (resp. $\mathcal{N}_{i+1}$) of $\mathbf{c}_i$ (resp. $\mathbf{c}_{i+1}$) such that any pair of points ($\mathbf{c}_i^*$,$\mathbf{c}_{i+1}^*$) belonging to $\mathcal{N}_i \times \mathcal{N}_{i+1}$ can be linked by an admissible parabola $\mathcal{P}_i^*$. Because Algorithm 5 tends to sample the environment uniformly, the probability of sampling two points in $\mathcal{N}_i$ and $\mathcal{N}_{i+1}$ respectively tends to 1 when time tends to infinity. $\mathcal{N}_i$ and $\mathcal{N}_{i+1}$ can be arbitrarily small. As a consequence, $\mathcal{P}_i^*$ can be arbitrarily close to $\mathcal{P}_i$. Because $\mathcal{P}_i$ is away about ε from the obstacles, $\mathcal{P}_i^*$ is guaranteed to be collision-free. □

Note that, contrary to classic path planning, ballistic motions can occur between

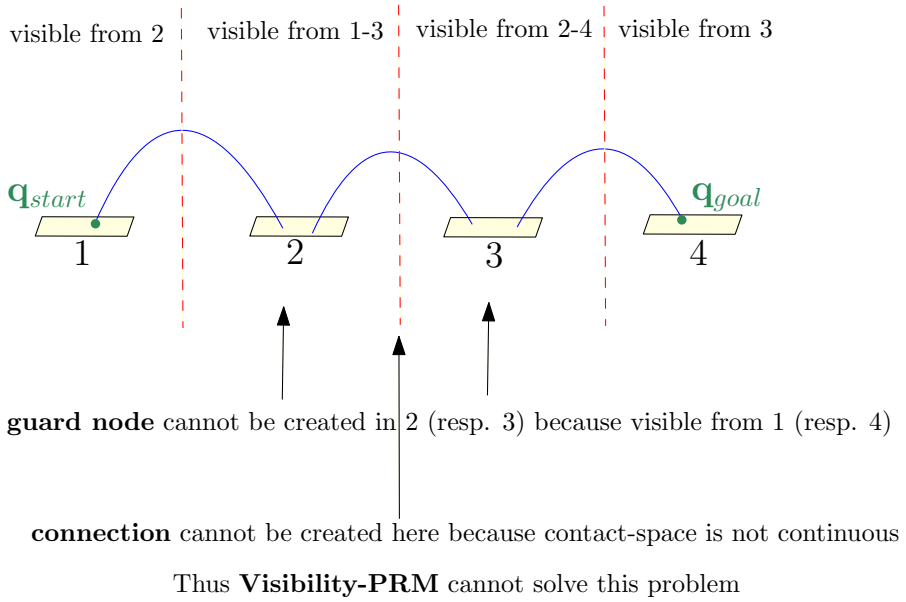


Figure 5.11: Example where Visibility-PRM will fail to find a path whereas a solution exists.

two non-continuous spaces, such as platforms, and the amplitude of a motion is limited by the takeoff velocity bound. Thus, completeness of planners using the ballistic steering method may become invalid if they require that paths lie in a continuous space. For example, the Visibility-PRM planner [Siméon 2000] builds a sparse roadmap contrary to PRM, based on nodes visibility to each other. Because nodes can only be created at some specific places of the environment, and the visibility is limited by the velocity constraint, the planner can become incomplete (see Figure 5.11).

5.5 Results

The ballistic motion planner was tested in 3D environments containing slippery surfaces, using HPP. Graphical renderings were done using Blender 2.7. In all described examples, the parameter n_{limit} from Algorithm 7 was set to 6.

We planned sequences of parabolas for a point-robot in three environments. For each example, we considered weak and strong constraints. The results are shown in Figures 5.12, 5.13 and 5.14. Movies of the trajectories are available in the companion video[1]. Solutions under strong constraints tend to increase the number of waypoints. It is not only a consequence of the velocity limitation that forces to reach closer positions (see also Figure 5.9 bottom), it also results from the cone narrowness. As it is shown in Figure 5.9 top, narrow cones provide parabolas with greater heights, more likely to produce collisions or to exceed the environment bounds.

[1] https://youtu.be/vv_K7HqANmk

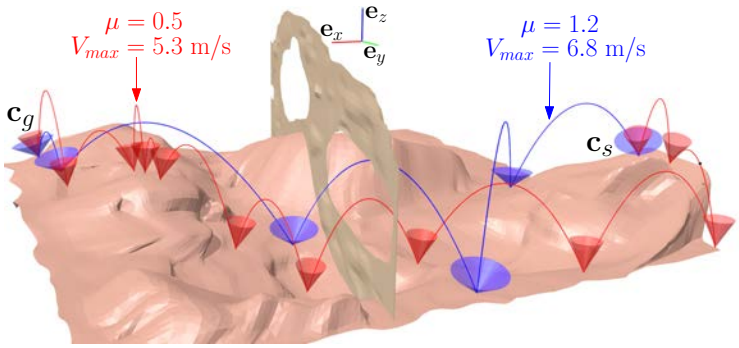


Figure 5.12: Full path planning results in an environment containing two windows to cross from right to left. The red solution is more constrained, so it results in a longer sequence of parabolas. Number of triangles: 47 733.

Parameters		Computation	Collision	Roadmap	Path
μ	V_{max}	time (s)	found	nodes	length (m)
0.5	6.5 m/s	9.89	3270	1995	39.9
0.5	7 m/s	9.99	4002	1835	37.4
1.2	6.5 m/s	1.04	601	282	28.1
1.2	7 m/s	0.909	540	237	27.0

Table 5.1: Averages of 40 ballistic planning of the example Figure 5.13, for four combinations of the parameters. Benchmarking was done on a PC with 4 GB of main memory and using one core of an Intel Xeon E3-1240 processor running at 3.4 GHz.

Table 5.1 presents the average performance results of the ballistic motion planner run on the Figure 5.13 benchmark. The velocity limitation is less restrictive in terms of computation time than the cone coefficient. However, the velocity limitation cannot be reduced without endangering the existence of a solution. In fact, the robot has to reach other platforms in order to find a solution path sequence.

5.6 Conclusions

We presented a method that analytically computes a non-sliding jump for a point-robot, resulting in a parabola going from one friction cone to another. The method has been implemented as a steering method in a probabilistic-roadmap motion planner in order to determine a sequence of jumps between given start and goal positions.

We believe that designing new type of paths, specially jumps, is relevant to improve the robot capacities to explore their environment. In particular, rescuing missions can rely on jumping to perform on rough terrains.

Besides, this method is the first stage of a more ambitious challenge. Our final purpose is to address dynamic motion planning for digital artifacts. The solution which we provide can be used to compute the center of mass path when the artifact is jumping. Now, it remains to consider more realistic models of contacts (e.g. mul-

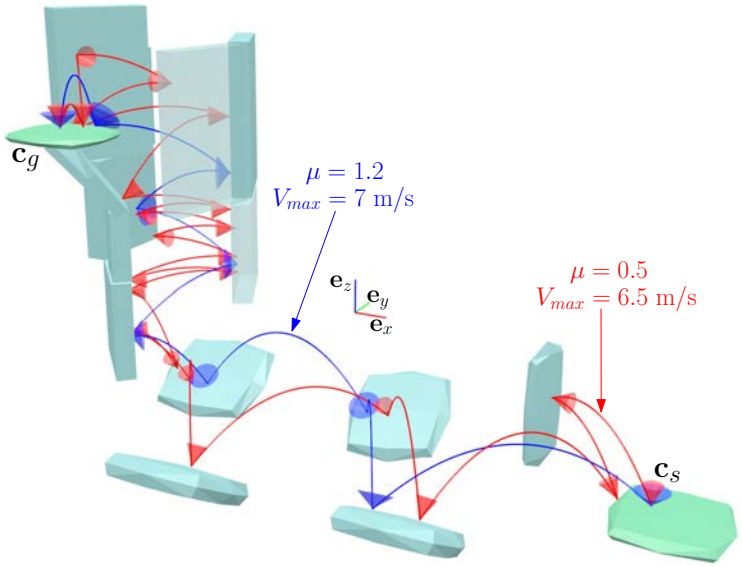


Figure 5.13: Full path planning results in an environment containing platforms and a chimney. The blue path is constrained by large cones, the red path by narrow cones. Reducing μ prevents the creation of a parabola linking two low platforms with the same inclination. Number of triangles: 696.

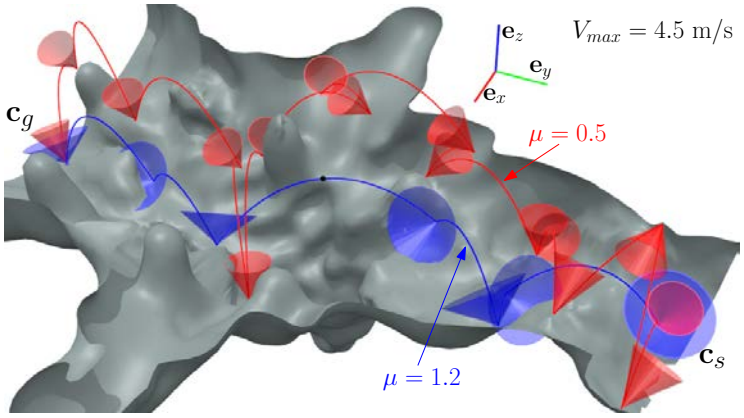


Figure 5.14: Path planning results in a cave. The robot has to avoid the numerous stalactites, stalagmites and holes. Number of triangles: 33513.

tiple contacts involving feet and hands) and impacts (e.g. including energy balance). Furthermore, the steering method we consider in the motion planner is assumed to be symmetric. This assumption is not realistic. Indeed, for a given parabola, the energy required to overcome the gravity effect from a position is greater than the energy to dissipate when landing at the same position. The extension of the motion planner to more realistic energetic models is the purpose of future developments.

Part of these perspectives is tackled in the next chapter, with the extension of the jumping motions to a wholebody character.

Chapter 6

Ballistic motion planning for jumping superheroes

Contents

6.1 Problem statement

Synthesizing high quality motions for legged characters in arbitrary environments is challenging:

(i) the dimension of the problem is high, equal to the number of DOFs,

(ii) the environment is big and complex, and

(iii) the motion is additionally constrained by the dynamics of the world.

For these reasons, motion synthesis is typically addressed with a decoupled approach. First, in the path planning phase a collision-free path is found for the character using a PRM-based approach [Kavraki 1996]. Then, in the animation phase a motion is computed and played along the path. Using this decoupled approach, we consider the issue of synthesizing highly dynamic jumping motions for legged characters. However in general there is no guarantee that a collision-free

path can be executed by a virtual character without considering a full dynamic model [Kunz 2014].

For concision, our scope is restricted to the generation of sequences of jumping motions, disregarding alternations with classic walking / running trajectories. Thus, given start and goal configurations for an character in an arbitrary environment, our framework outputs a motion described as a sequence of parabolic wholebody jumps, respecting a set of kinematic and dynamic constraints for the character.

The key idea is the introduction of a simplified multi-contact model within a sampling based planner. We relax the dynamics of the problem with an impulse hypothesis, which assimilates our character with a superhero. Our character is thus assumed to be able to exert a large force instantaneously, as proposed in Chapter 5. This formulation simplifies the verification of the Newton equation of the motion regarding the dynamic constraints on the character. We assume that the Euler equation is always satisfied as we consider centroidal dynamics.

The impulse hypothesis leads to an efficient, low dimensional formulation of the motion planning problem, solved with a sequence of simple geometric tests, while partially capturing the dynamic model of the character to compute plausible trajectories. To handle the combinatorial aspect of the contact generation problem, we decouple the trajectory planning phase from the contact generation phase, thanks to a heuristic based on the reachable workspace of the character [Tonneau 2015b]. In this work, the authors introduce a way to decouple the character model into a dual-shape including the limb reachable spaces. Thus, planning with this reduced system is faster while obstacle reachability for contacts is verified. Once contact placements are chosen, classic non-sliding constraints are transferred from the contact positions to the character center of mass (COM) through a new friction cone representation which complies with the use of the Ballistic Motion Planner.

Finally, to propose a complete framework, we implement a method based on key-frame interpolation to automatically animate the computed trajectory. Some of the key-frames are generated automatically by the method while others are predefined by the user. Since motion generation may produce limb collisions or contact-constraints violation, a local planner is used to prevent these effects.

6.2 Non-slipping constraint for an arbitrary number of contacts

We denote by $m \in \mathbb{R}$ the mass of the character or robot. The COM position $\mathbf{c} \in \mathbb{R}^3$ depends on the posture of the system. It is not a fixed point belonging to the body. However, for simplification purpose, we consider the center of mass as lying in the body-root of the kinematic chain. $\dot{\mathbf{c}} \in \mathbb{R}^3$ is the velocity of $\mathbf{c}$ and $\ddot{\mathbf{c}} \in \mathbb{R}^3$ is its acceleration.

Our contact model is based on Coulomb's non-slipping constraint, which we recall briefly in this section. We generalize the constraint to handle an arbitrary number of contacts, by expressing it at the COM of the robot. From this formu-

lation, assuming an impulse formulation of the model, we propose a conservative condition for non-slipping, that is sufficient, but not necessary. This condition, reduced to a simple geometric test, is then used in our motion planner, presented in Section 5.3.

For the i-th contact point $\mathbf{p}_i$, $1 \leq i \leq h$, $\mathcal{K}_i$ is the associated convex friction cone, considering a friction coefficient μ_i, and a surface normal $\mathbf{n}_i$. $\mathbf{f}_i$ is the contact force applied at $\mathbf{p}_i$. The non-slipping condition is still given by Coulomb's law: the contact will not slip if the contact reaction force $\mathbf{f}_i$ lies strictly within the friction cone $\mathcal{K}_i$.

Now, if we consider an arbitrary number h of contacts, each reaction force $\mathbf{f}_i$ must lie in its associated cone $\mathcal{K}_i$. The resulting force $\mathbf{f_c}$ applied at the center of mass $\mathbf{c}$ of the character, is defined as the sum of all the forces $\mathbf{f}_i$. It follows that the set of admissible resulting forces $\mathcal{K}$ such that the non-slipping condition is respected is defined as the Minkowski sum of each individual friction cone:

$$\mathcal{K} = \{\mathbf{f}_1 + \cdots + \mathbf{f}_h | \mathbf{f}_i \in \mathcal{K}_i\} \tag{6.1}$$

As a Minkowski sum of convex cones, $\mathcal{K}$ is itself a convex cone [Boyd 2004]. Since the analytical form of $\mathcal{K}$ is unknown, it is common to process the Minkowski sum of the linearized shape of the cones [Bretl 2008]. However, linearization is conservative and it introduces noise. For instance, in the work of [Carón 2016], the method based on *cdd* cannot afford three or more contact supports in terms of computation time. Instead, we apply the fact that what we need for our path planner is the intersection of $\mathcal{K}$ with a vertical plane including the parabola trajectory. In such a context, an analytic computation of the intersection can be simply provided, rather than using a linear approximation of the cones. The details of this intersection computation are given in Appendix A Section A.1. Due to the vertical plane dependency, the planner modification will be detailed in a further section. The resulting shape, a 2D cone or a point, is denoted $\mathcal{K}_\mathbf{c}$, originating at $\mathbf{c}$, of normal $\mathbf{n_c}$ and friction $\mu_\mathbf{c}$. Figure 6.1 illustrates the construction of $\mathcal{K}_\mathbf{c}$ in the case of two and three contact points.

The cone $\mathcal{K}_\mathbf{c}$ is included in $\mathcal{K}$ by construction, and is thus a conservative approximation of $\mathcal{K}$. Force closure contact configurations, where the resulting normal $\mathbf{n_c}$ is null, are considered invalid in this formulation. Although this formulation is intuitively really conservative, it has an analytic form that makes it extremely efficient to compute. We justify the interest of this formulation with the variety of the solutions found by our planner.

For the impulse-force model of the non-sliding constraints, we simply extend the model proposed in Section 5.3.1 to multi-contact motions, by applying it to the centroidal cone $\mathcal{K}_\mathbf{c}$.

Let us assume that between two jumps, there exists an impulse-force that instantaneously changes the COM velocity from landing to next takeoff. By the propriety of the convex cone, this implies that there exists a distribution of impulse-forces at each contact point which belong to their respective friction cones. The force applied

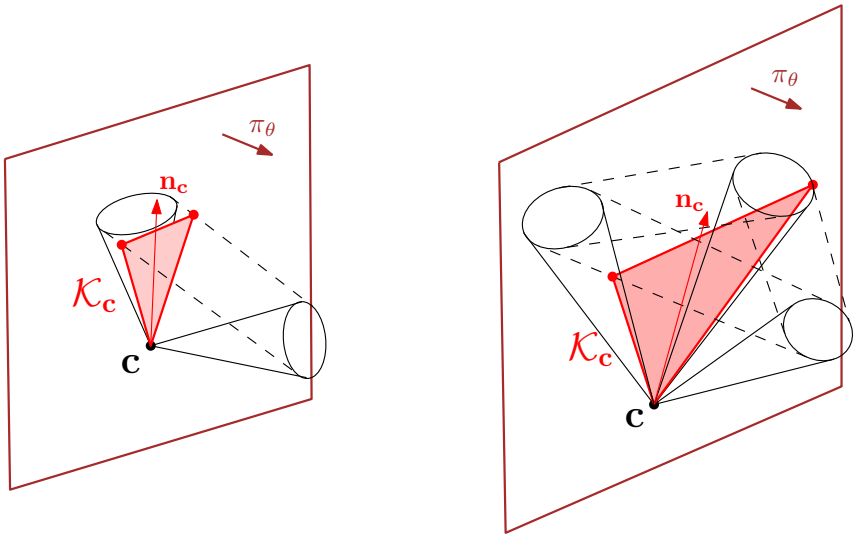


Figure 6.1: Two examples of intersections between a convex-cone and a vertical plane, for two (left) and three (right) summed cones.

to the COM is then the resultant of the contact forces. Therefore, the combination of the impulse-force model and the convex-cone is dynamically valid with the friction constraints.

6.3 A reduced character model for contact location estimation

Motion planning for legged characters requires generating contact configurations for force exertion. This is hard because it is impossible to generate randomly a contact configuration. Random contact configuration are obtained by selecting randomly a collision-free configuration and by projecting it on the boundary of an obstacle. Thus projectors are required [Bretl 2004]. They mostly consist in iterative projections that solve the inverse kinematics, which is time consuming. Furthermore, contacts are associated with kinematic and dynamic constraints, and introduce combinatorics hard to handle for such approaches, resulting into hours of computation. In order to reduce the dimensionality of the problem and break the combinatorial complexity, the planner does not consider neither the complete character model at this phase, nor the explicit computation of contact configurations.

We consider a legged character, described by a kinematic chain R, composed of a root R^0, and l limbs $R^k, 1 \leq k \leq l$. The root has a minimum of $r \geq 6$ degrees of freedom (DOFs), which describe its position and orientation in the world frame. The additional DOFs describe the articulations the character torso, head, spine etc. R is fully described by a configuration $\mathbf{q} \in \mathbb{R}^{r+n}$.

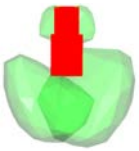
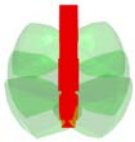
Character	Fullbody representation	Reduced representation	Full number of DOFs	Reduced number of DOFs	Friction coefficient	Maximal takeoff - landing velocities (m/s)
Skeleton			43	15	0.6	8 - 12
Jumper-man			38	12	1.2	10 - 15
Frog			45	9	1.2	6 - 8
Ant			66	12	1.2	4.5 - 8
Lamp			11	6	1	8 - 9
Kangaroo			42	18	1.2	7 - 8

Table 6.1: Character models[1] and default planning parameters. The **W** shapes of each character are shown on the right. **W** is composed of the trunk and the accessibility workspaces of the limbs (green). The trunk is approximated with bounding boxes (red). Spine DOFs of the Skeleton are not activated because they can be neglected [Hickox 2016].

We define:

- $\mathbf{q}^k$ denotes the configuration (a vector of joint values) of the limb R^k;
- $\mathbf{q}^{\overline{k}}$ denotes the vector of joint values of R **not** related to R^k. We note for convenience $\mathbf{q} = \mathbf{q}^k \oplus \mathbf{q}^{\overline{k}}$;
- $\mathbf{q}^0 \in \mathbb{R}^r$ denotes the configuration of the root R^0.

At the planning phase, to check the non-slipping condition, we need an estimation of the contact locations. To avoid dealing with the combinatorial and computational complexity of contact generation, we introduce a contact estimation heuristic, based on a dual, low-dimensional representation of the character, introduced by [Tonneau 2015b]. We recall it here for completeness.

[1] 3D models are freely available and can be found in the following websites: http://tf3dm.com/ and http://archive3d.net/.

Rather than considering the complete body configuration, the planner only considers the root configuration $\mathbf{q}^0$. To perform collision detection, the complete body is approximated with a bounding shape $\mathbf{W}^0$. Additionally, for each limb k, we attach to the root a shape $\mathbf{W}^k$, computed as the reachable workspace of the limb. Its computation is based on gathering the end-effector positions for numerous random configurations of the limb (typically 10 000 samples). Then, a convex envelop of these samples is generated as the $\mathbf{W}^k$ shape. The list of character models and parameters is presented in Table 6.1. Note that the friction coefficients could also depend on the environment materials, but for convenience we assume that they are constant for each character.

Our contribution is to use the reachable workspace of each limb for contact location estimation. Given a configuration $\mathbf{q}^0$, we assume that if $\mathbf{W}^k$ is in collision with the environment, it is possible for the character to create a contact (Figure 6.3) between the limb R^k and the environment. The contact location is estimated to be roughly at the center of the intersection between the largest colliding contact surface and $\mathbf{W}^k$. Figure 6.2 presents examples of contact locations.

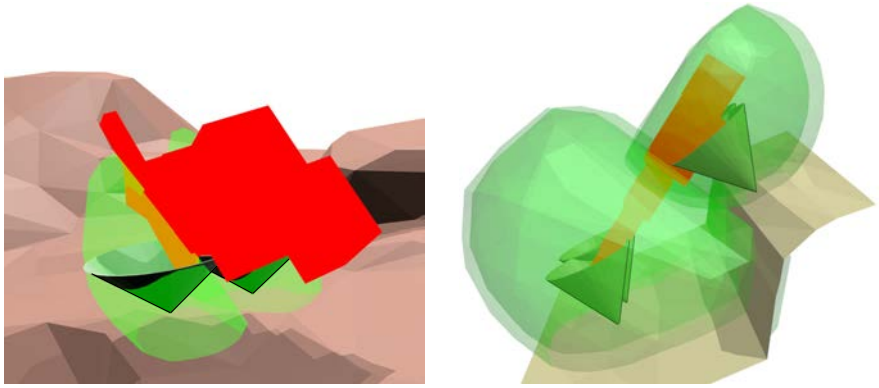


Figure 6.2: Two examples of contact cone locations from the intersections between reachable workspaces $\mathbf{W}^k$ (green) and environments.

This heuristic provides an efficient method to approximate the contact location, and allows the verification of the non-slipping condition without considering the expensive complete model (i.e. no limb configuration is computed). The contact-configuration generation step is detailed in Section 6.5.1.

6.4 Motion planning algorithm for the reduced model

The animation-planning procedure is divided in two main steps: planning the path $\bar{\mathbf{s}}$ of the dual-shape and then animating it into a trajectory $\mathbf{s}$. This section deals with the first planning step. Note that animation is not directly processed during the planning stage due to performance considerations.

Our algorithm is an extension of the Ballistic Motion Planner introduced in Chapter 5, adapted to generate multi-contact jumping motions. We modify the original algorithm in three ways:

- First, as proposed by [Tonneau 2015b], the configuration sampling is biased towards configurations that allow to generate contacts. Without loss of completeness, we only consider configurations for which at least n_{cn} shapes $\mathbf{W}^k$ are in collision, i.e. n_{cn} contact creations are possible (Figure 6.3). n_{cn} is typically equal to the number of shapes $\mathbf{W}^k$, except for humanoids where $n_{cn} = 2$;

- Then, to generate the root trajectory between two configurations, we use the parabolic steering method previously introduced to represent the root location. Other root DOFs such as orientation are randomly generated. As we limit the takeoff and landing velocity magnitudes by different values, paths are now oriented;

- Our contribution lies in the validation of the generated trajectory. A trajectory between two configurations is only validated if the multi-contact non-slipping condition is validated.

First, given an environment, a roadmap can be generated to capture the topology of the space regarding the reduced robot (preComputeRoadmap). A user-defined termination condition typically determines the duration, or number of iterations, of the roadmap exploration. Once the graph has been generated, requesting a trajectory between two given configurations $(\mathbf{q}^0_{start}, \mathbf{q}^0_{goal})$ consists in adding them to the roadmap using Algorithm 8. Each requested configuration is assumed to be close enough to obstacles to perform contacts if needed. If both configurations have been successfully added, the shortest trajectory connecting them is obtained by computing the shortest traversal of the graph.

We detail the algorithm functions:

- validTrunkRandomSample returns a root configuration $\mathbf{q}^0$ such that $\mathbf{W}^0$ is collision-free, and at least n_{cn} number of shapes $\mathbf{W}^k$ are in collision. Computation details of $\mathbf{q}^0$ can be found in Section 6.2.1 of [Tonneau 2015a]. Instead

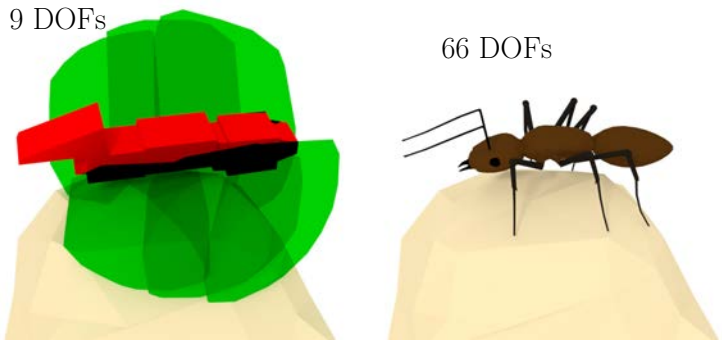


Figure 6.3: Illustration of a necessary condition for contact creation. If the trunk bounding box is collision-free (left–red), and the reachable workspaces of the limbs are in collision with the environment (left–green), we assume that a contact configuration can be created between the effectors and the environment (right).

Algorithm 8 Ballistic motion planner for a **W**-shaped system.

Input: $\mathbf{q}^0_{start}$, $\mathbf{q}^0_{goal}$, *Environment*, μ, $\dot{\mathbf{c}}^{max}_l$, $\dot{\mathbf{c}}^{max}_t$
Output: Sequence of jumps linking the two configurations

```
Roadmap ← preComputeRoadmap()
findContacts(q_s^0)
K_c ← computeCone(q_s^0)
addToRoadmap(q_s^0, K_c)
findContacts(q_g^0)
K_c ← computeCone(q_g^0)
addToRoadmap(q_g^0, K_c)
connected ← False
while not(connected) do
    q_rand^0 ← validTrunkRandomSample()
    findContacts(q_rand^0)
    K_c ← computeCone(q_rand^0)
    addToRoadmap(q_rand^0, K_c)
    for q^0 ∈ Roadmap do
        T ←validSteer(q^0, q_rand^0, ċ_l^max, ċ_t^max)
        addToRoadmap(T)
        T ←validSteer(q_rand^0, q^0, ċ_l^max, ċ_t^max)
        addToRoadmap(T)
    connected ← areConnected(q_s^0, q_g^0)
s̄ ← findShortestPath(q_s^0, q_g^0)
s̄ ← rotateAlongPath(s̄)
return s̄
```

of being random, the trunk orientation can intuitively be modified. For instance, the trunk is orientated so that the robot lies on the ground (standing or on all fours). This prevents orientations where creating a contact is possible but unnatural to provide an impulsion (e.g. a hand in the back of the character pushing a wall).

- findContacts estimates the contact positions on the surfaces resulting of the intersections between the $\mathbf{W}^k$ shapes and the environment. From the estimated contact locations, the centroidal cone $\mathcal{K}_\mathbf{c}$ is computed with the method computeCone.

- validSteer is an extension from the parabola-steering-method of Chapter 5. To generate a trajectory between two configurations $\mathbf{q}^0_a$ and $\mathbf{q}^0_b$, the method determines whether there exists a parabola that verifies the non-sliding condition. This means that the takeoff velocity belongs to the centroidal cone $\mathcal{K}_\mathbf{c}(a)$, while the landing velocity belongs to $\mathcal{K}_\mathbf{c}(b)$. The takeoff and landing velocities are also limited by user-defined bounds $\dot{\mathbf{c}}^{max}_t$ and $\dot{\mathbf{c}}^{max}_l$. The trajectory is validated if the resulting parabola is collision-free, that is if $\mathbf{W}^0$ is collision-free along the path. If no valid parabola can be found, validSteer

returns an empty trajectory $\mathbf{T}$.

- TerminationCondition is a user-defined function that determines the duration, or number of iterations, of the roadmap exploration. In the case of a planning query between two configurations, this function is replaced by areStartAndGoalConnected which indicated if the configurations are linked through the roadmap.

- findShortestPath builds the sequence of parabola trajectories, using classic A* algorithm that searches for the shortest path through the roadmap to connect the two given configurations.

- rotateAlongPath is a re-orientation of the character trunk to follow the next parabola-direction, while the robot still lays on the ground as anticipated during the shooting stage. The re-orientation method is a personalized effect to improve realism. During the sequence of jumps, the character turns so that at the end of a jump, it faces the next jump direction. As a consequence, the character is not rotating during the last jump (Figure 6.4). If the rotation effect is invalidating a transition configuration, the orientation given by validTrunkRandomSample is conserved as it is generated to be valid (see Figure 6.5).

Computation details of re-orientation procedures are given in Appendix B.

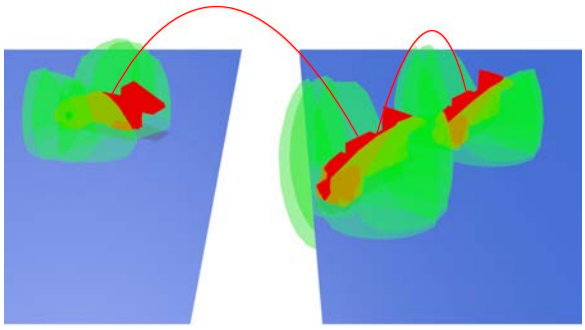


Figure 6.4: Sequence of parabola paths for an ant reduced model. The takeoff velocity limitation prevents the direct connection from start to goal, so a waypoint is found by the planner to solve the problem. At each waypoint configuration, the accessible workspaces of the legs are in collision with the environment. During the first jump, the robot is rotating to match the second parabola orientation.

6.5 Motion synthesis for wholebody animation

A remaining step is to extend the trajectory $\mathbf{q}^0(t)$ computed for the root of the character into a full body animation. In this phase, despite the impulse model formulation, to obtain a plausible animation, we consider that contact duration is

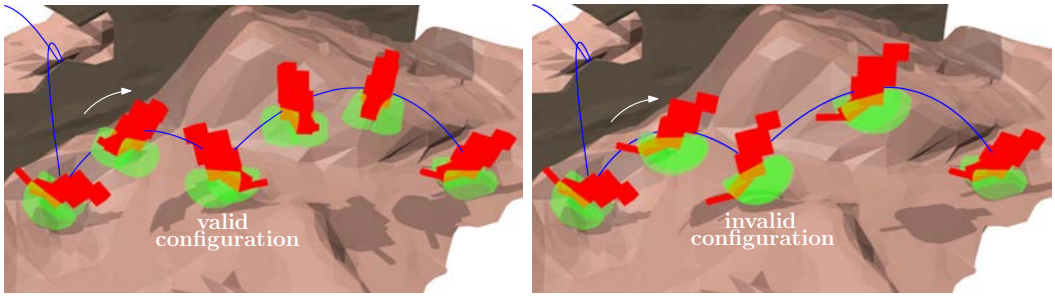


Figure 6.5: Comparison of orientations along a path of the reduced kangaroo in a desert. (Left) The original path contains a randomly generated configurations so the interpolated orientation may appear as unnatural. (Right) The random configuration is re-orientated to fit the path direction. However it becomes invalid as the $\mathbf{W}^k$ shapes are no longer in collision with the ground. Therefore the new orientation is not retained.

not instantaneous. This requires identifying the transition times between contact and flight for each effector.

The wholebody animation technique proceeds as follows for each jump:

- First, several key configurations are automatically computed at specific times of the trajectory: two contact configurations at the takeoff and landing phases; one configuration at each moment where an effector contact is broken (respectively created); one configuration at the apex of the trajectory (Figure 6.10).

- Then, a linear interpolation is performed for each DOF of the character between each key configurations. It is designed in such a way that collisions are avoided and contact location constraints are maintained.

6.5.1 Computation of wholebody contact configurations, and identification of takeoff and landing phases

We first address the identification and animation of the takeoff and landing phases, where contacts occur between the character and the environment. To keep the explanation simple, in this section we consider an input trajectory $\mathbf{q}^0(t)$ which consists in a single jump, and assume that contacts are created and broken simultaneously, although the method handles the general cases.

We first generate a collision-free, wholebody contact configuration that verifies the non-sliding condition at the initial frame (which corresponds to the exact parabola extremity) $\mathbf{q}^t_{contact}$. This configuration is generated using Inverse Kinematics (IK), where the effector is placed on the obstacle surface with eventually a constraint on its orientation (e.g. the hand is *facing* the surface). To increase the plausibility of the contact configuration, a heuristic is used to bias contact generation towards configurations as close as possible to a pre-defined reference configuration (Figure 6.6). Contact locations may differ from the planned ones since if a re-orientation was applied to the waypoint, the contacts may no be reachable

anymore by the respective limbs. Furthermore, configurations returned by IK may not be satisfying in terms of realism for takeoff and landing postures.

New contacts are accepted as long as the corresponding centroidal cone $\mathcal{K}_{\mathbf{c}}$ is compatible with the trajectory (i.e. satisfies the criterion (5.7) of Chapter 5). An example is provided in Figure 6.7.

For visual purpose, contacts are not instantaneously released when the character takes off from $\mathbf{q}^t_{contact}$. Instead, we identify the transition time between the takeoff and flight phases using an iterative approach. We go forward in time from $t_0 = 0$. Next iterations are computed as $t_{i+1} = t_i + \delta t$. δt is set to 5 ms to avoid breaking the IK. At each time step t_i, we update the root configuration $\mathbf{q}^0(t_i)$, and solve an inverse kinematics problem for each limb to maintain the contacts active at the same locations than initially with the obstacle surface. If possible, the end-effector initial orientations are conserved as well. The last time $t_{transit}$ before the inverse kinematics fails is the transition time with the flight phase. The corresponding wholebody configuration is denoted $\mathbf{q}^t_{transition}$. Note that contacts can be released at different times, $t_{transit}$ is computed when the last contact could not be maintained. Different termination conditions are also included, such as a maximal number of iterations (typically 100), or a ratio of the parabola length (e.g. not further than on third of the parabola length). Finally, the landing phase is handled similarly, with the exception that we go backwards in time from the impact time, and the configuration is denoted $\mathbf{q}^l_{transition}$.

For the Jumper-man character, Figure 6.8 illustrates two key-frames, the initial configuration and the last configuration of the landing phase before contacts are released. Arms configurations were designed to increase the motion plausibility when they do not create contacts with the environment.

Note that other heuristics can be used to generate the contact configuration, instead of bias it toward a reference posture. The two following methods can be considered:

- The EFORT heuristic provides the best contact configuration given a force direction to process a motion toward this direction [Tonneau 2015b]. Based on the resultant of the contact-force applied to $\mathbf{c}$, EFORT can lead to relevant

Figure 6.6: Illustration of the generation of a contact configuration. (Left) The manually defined reference configuration. (Middle) The root of the reference configuration is placed close to an obstacle so that the obstacle is in the accessible space of the limbs. (Right) Result of the configuration projection on the obstacle to create contacts with the hands and the feet.

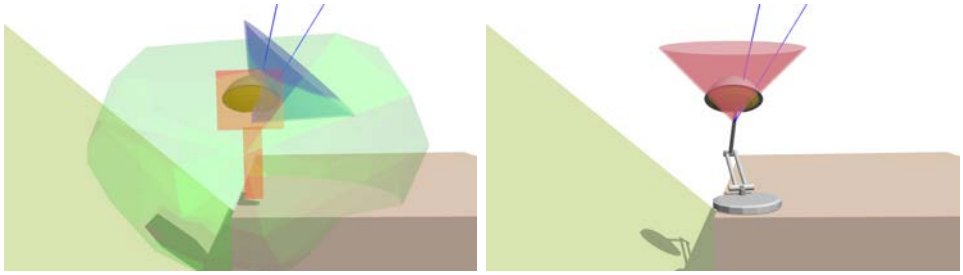


Figure 6.7: Different contact cones for the jumping lamp. (Left) The blue cone comes from the intersection between the $\mathbf{W}^k$ shape and the left surface, and is moved to the COM of the reduced lamp model. It is used to validate the non-sliding constraint during the planning stage. (Right) The red cone is the friction cone from the contact on the right surface, moved to the COM. As the parabola lies inside the red cone, this configuration satisfies the non-sliding constraint.

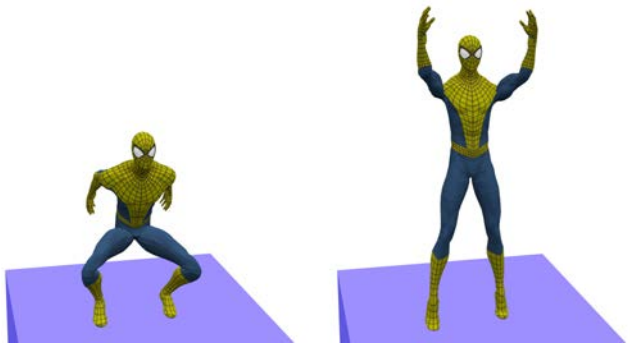


Figure 6.8: Key-frames of Jumper-man during a contact (left) or during a transition to a flight phase (right). These configurations increase the plausibility of the takeoff and landing motions when arms are not constrained to be in contact with the environment.

results.

- A manipulability heuristic returns the best configuration to process a motion toward any directions. This option is more accurate if no force-direction is provided, e.g. the two paths which arrives to and goes from the contact pose are unknown.

Previous methods are interesting in the case where the user cannot define a reference configuration. A qualitative comparison between contact configurations from the reference-configuration heuristic and from EFORT is illustrated in Figure 6.9. As the heuristic choice is only an appearance criterion, we choose to rely on the reference-based one.

6.5.2 Wholebody animation of the jump trajectory

For the flight animation, the root motion is guaranteed to be collision-free and remains unchanged. The limb configurations are not simply interpolated between the contact configurations ($\mathbf{q}^t_{contact}$, $\mathbf{q}^l_{contact}$) because it looks unnatural. Instead, for

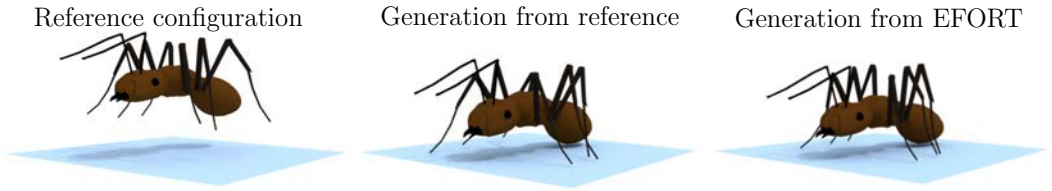


Figure 6.9: Contact configuration examples of the ant character. (Middle) The heuristic based on a reference configuration is used. (Right) EFORT with an upward direction is applied.

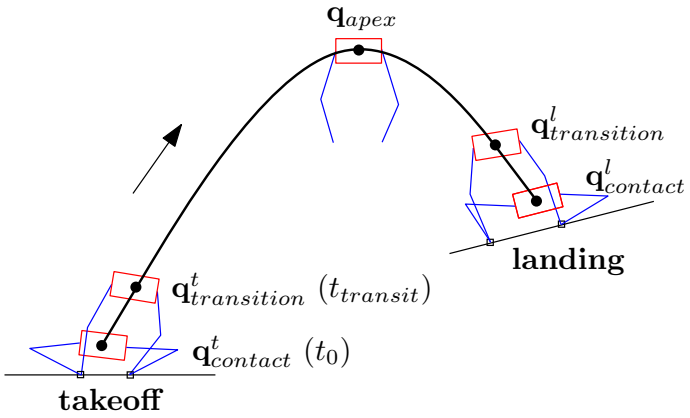


Figure 6.10: Key configurations computed along a parabola trajectory.

each parabola path, we first insert three key-frame limb configurations, including the transition contact configurations ($\mathbf{q}^t_{transition}$, $\mathbf{q}^l_{transition}$) of the previous section. An additional key configuration is generated at the apex of the parabola $\mathbf{q}_{apex}$. It is sampled to be collision-free, with a bias to lie in the neighborhood of a manually defined reference configuration. Key-frames positions on a parabola path are illustrated in Figure 6.10.

Then, the linear interpolation between the key-frame configurations can result in one limb being in collision with the environment or another limb. This situation is equivalent to a local motion planning where a collision-free path has to be found for the limb between its two key-configurations. To solve it, we simply use the sample-based planner bi-RRT. During this re-sampling, only the DOFs of the limb in collision are modified.

Note that no solution to limb collisions may be found. In that case, forcing the framework to return a valid solution may output from a trade-off between collision avoidance and realism. For example, problematic limb trajectories may be frozen to one configuration which reduces the risk of collision with other limbs or the environment. Contacts may also be canceled to more easily find collision-free trajectories.

A summary of the wholebody configuration generation and the flight interpolation is provided in Algorithm 9.

Algorithm 9 Wholebody animation procedure from a root trajectory with manually defined reference limb configurations.

Input: *environment*, $\mathbf{q}^0(t)$, *reference configurations*
Output: Animated path $\mathbf{s}$

```
[q^t_contact, q^l_contact] ← GENERATECONTACTS(s̄)
for each parabola p̄ ∈ s̄ do
    do
        q^t_transition ← MAINTAINCONTACTS(p̄, q^t_contact)
        q^l_transition ← MAINTAINCONTACTS(p̄, q^l_contact)
    while (MAINTAINED(q^t_contact, q^t_transition) or MAINTAINED(q^l_contact, q^l_transition))
    p ← INTERPOLATEANDSOLVE(p̄, q^t_contact, q^t_transition, q_apex, q^l_transition, q^l_contact)
    ADDTOSEQUENCE(p, s)
return s
```

6.6 Simulations

Our framework has been implemented in HPP. Final renderings have been generating with Blender 2.7 using the automatic blender export tool of HPP.

We demonstrate the genericity of our approach with four characters in five environments. The characters have different morphologies from two up to six limbs. They also have different jump abilities, expressed as the friction coefficients of the environment, as well as velocity bounds given in Table 6.1. We first detail key qualitative results on the different scenarios that illustrate the aspects and the limits of the method. Details are also available in the companion video[2]. Then performances are provided and discussed concerning the real time potential of the approach.

6.6.1 Qualitative results

Jumper-man on cubes This example justifies the use of the non-sliding constraint to eliminate unnatural jumps (Figure 6.11). Without friction cones to constraint the parabolas, the COM trajectory is tangent to the obstacle surface. In the video, we provide another example of a jumping ant without sliding constraints.

Jumper-man on houses This scenario involves the Jumper-Man character jumping on house roofs (Figure 6.12–red). This example shows the genericity of our approach, which handles well cases where assumptions on the number of contacts are not verified, due to the approximations made by our planner. In Figure 6.13, for instance, the jump transition only occurs with one foot, even if both leg accessible spaces were in collision with an obstacle. This case shows that we can create an arbitrary number of contacts (two or one here) during the trajectory, where motion capture based classic approaches would fail. In Figure 6.14, we provide another situation where Jumper-man is jumping on cubes from two to four contacts.

[2]Video of the simulations: https://youtu.be/GGisCV5BoHw

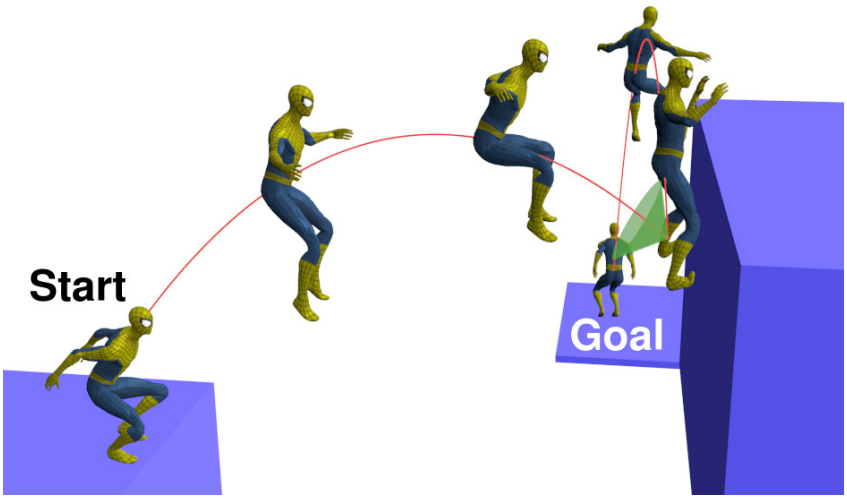


Figure 6.11: Trajectory of Jumper-man in the cubic environment, where non-sliding constraints are disabled. Friction cones are only displayed to help understanding the trajectory. As constraints are off, the second parabola that is outside of the friction cone is accepted by the planner. However, as it is tangent to the obstacle surface, the second jump seems unrealistic.

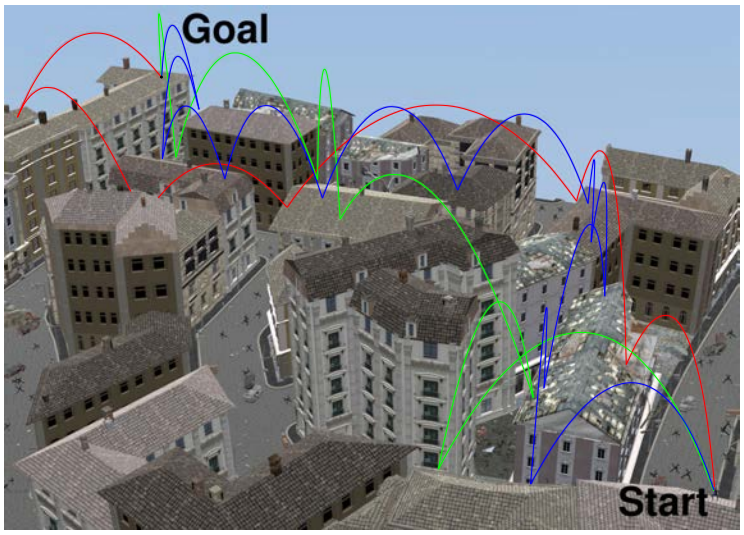


Figure 6.12: Trajectories of Jumper-man (red) and the skeleton (blue and green) on the roofs. The skeleton has the same velocity limits (25 m/s) as Jumper-man for the green curve, and reduced ones (18 m/s) for the blue curve.

Skeleton on houses We consider the same environment, initial and final configurations, addressed with a different character. The previous path cannot be the same because we decrease the friction coefficient in first case (Figure 6.12–green). The number of jumps in the resulting trajectory does not change (6) because the

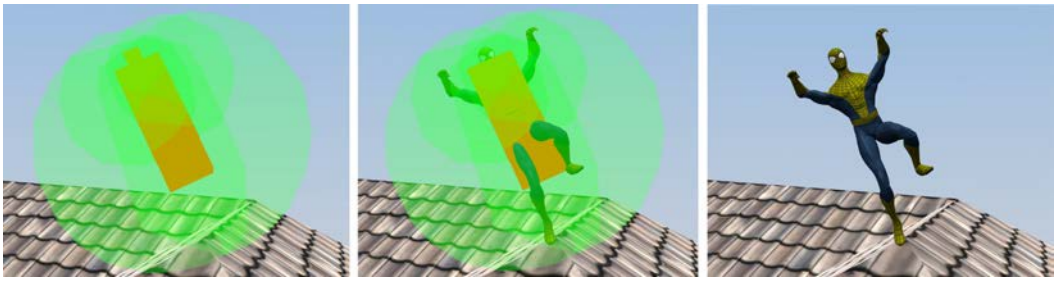


Figure 6.13: During the path planning of the Jumper-man on roofs, one reduced configuration is found on a roof edge (left). Both leg accessible spaces are in collision so the reduced configuration is valid. However, while generating contacts, the right foot is firstly placed on the only valid spot for the left foot (right). Our method handles this case correctly despite a wrong initial assumption on the number of contacts, contrary to existing approaches.

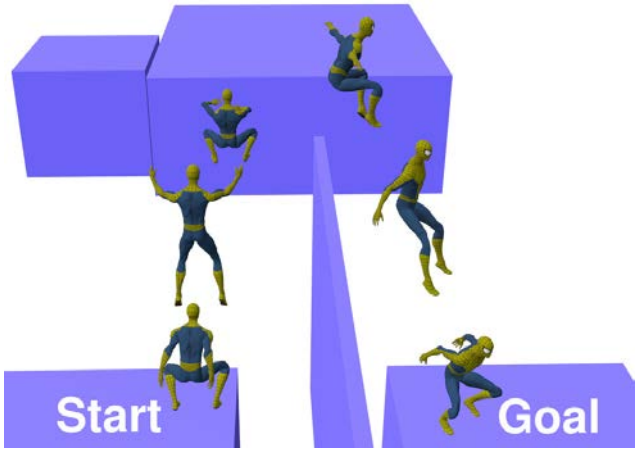


Figure 6.14: Jumper-man jumping on cubes, with a transition occurring on four contacts.

planned path is more direct towards the goal than the Jumper-man path which is making a detour. Then we decrease the takeoff velocity bound such that it is impossible to do a very long jump. This results in an increase in the number of jumps, to 11. (Figure 6.12–blue).

Skeleton in desert This environment contains a narrow passage that justifies the need of the motion planner. The character has to find one of the two possible holes to cross the wall. It also has to jump high enough to prevent collision between its legs and the wall bottom. Results are shown by Figure 6.15. Despite the non-sliding constraints, some jump transitions are in the limits of the friction cone. Thus the character has to process an important angular momentum to execute its motion.

Frog in pond We present a frog jumping on rock and plants in a pond-environment (see Figure 6.16). This example accounts for the genericity of our method.

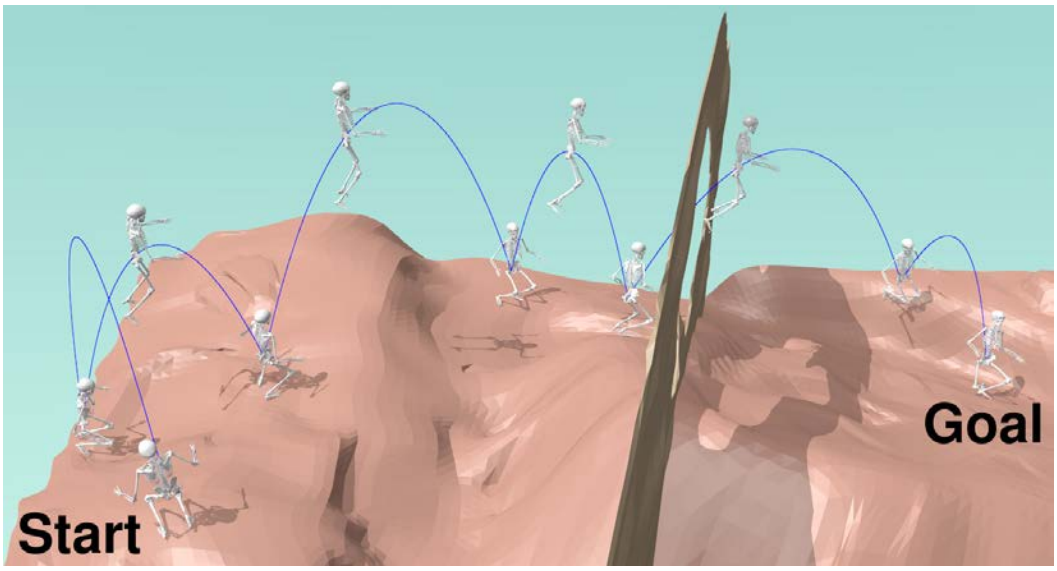


Figure 6.15: Trajectories of the skeleton in a desert with a holed wall.

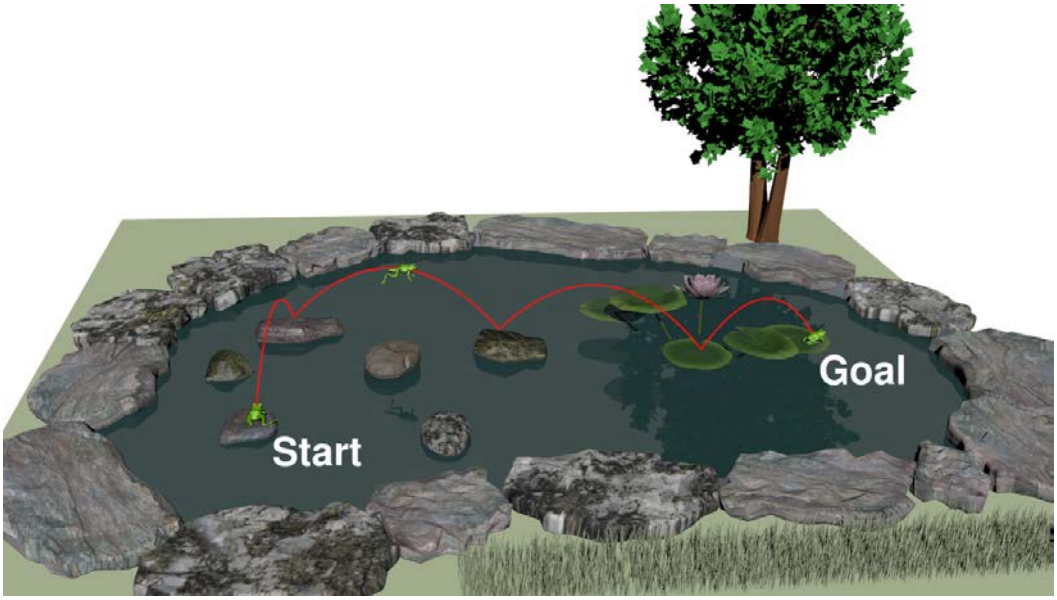


Figure 6.16: Full trajectory of the frog character in the pond environment.

Lamp on platforms This example involves a lamp jumping on platforms (see Figure 6.17). This is the most simple example in terms of contact creation and non-sliding verification since the character has only one limb. However, going down by jumping on the vertical walls seems demanding on the angular momentum generation. Using our heuristic rejecting implausible momentum generation, we can invalidate such trajectories.

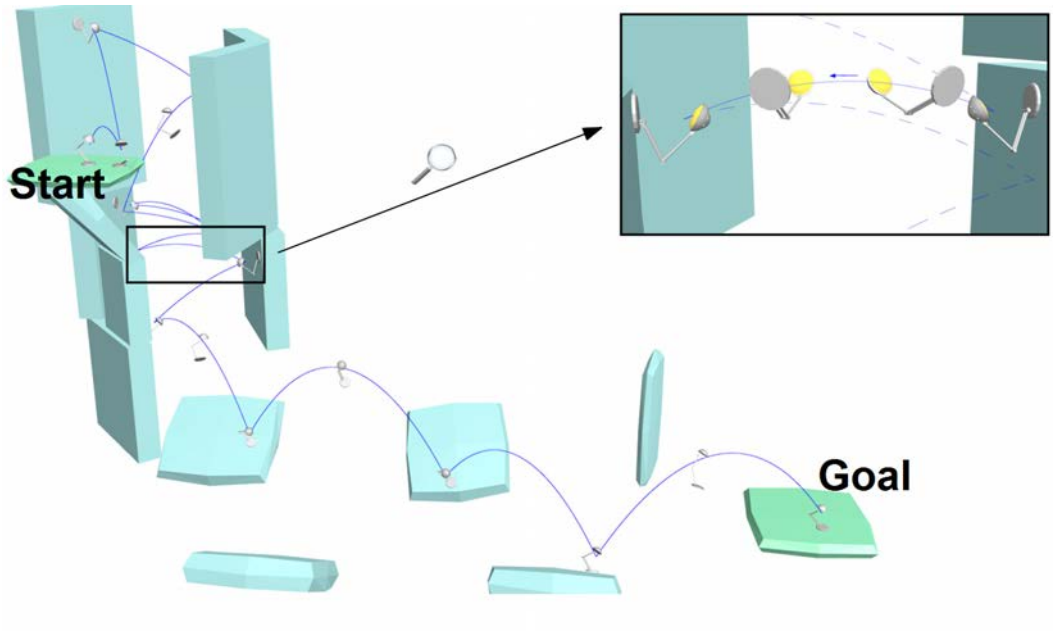


Figure 6.17: Shapshots of the jumping lamp trajectory on platforms. On top-right, zoom on a transition between two vertical walls.

6.6.2 Time performances

Scenarios	Offline		Online				
	Roadmap CT (s)	N. of nodes	Full trajectory CT (s)	One-jump CT (s) Average	Min | Max	N. of jumps	Contact generation (s)
Skeleton in desert	0.51	22.4	2.8718	0.3231	0.2167 - 0.9147	9.2	0.0253
Frog in pond	0.76	46.6	0.4742	0.1185	0.1077 - 13.25	5.3	0.0326
Jumper-man on houses*	44.93	142.4	7.2497	1.3894	0.3307 - 3.9250	6.1	0.0765
Skeleton on houses*	107.13	1256.3	7.2737	0.5057	0.4381 - 0.6923	14.4	0.0264

Table 6.2: Computation time (CT) averages of the method stages for 50 runs of each of the scenarios. Offline columns concern the planning procedure while online columns deal with the path query in the roadmap and the motion synthesis of the whole trajectory. The number of nodes reflects the roadmap memory occupation. Number of triangles: house 449267, desert 47733, pond 6994, Jumper-man 10052. *Skeleton parameters: $\dot{\mathbf{c}}_t^{max} = 18$ m/s, $\dot{\mathbf{c}}_l^{max} = 25$ m/s. *Jumper-man parameters: $\dot{\mathbf{c}}_t^{max} = \dot{\mathbf{c}}_l^{max} = 25$ m/s.

Table 6.2 provides a quantitative analysis of our approach performances in four scenarios. We separate the offline step, including the roadmap construction, from

the online step, including the trajectory query and the jump synthesis. Since a full-path motion synthesis is dependent on the number of waypoints found by the planner, we give the average computation times to synthesize one jump and one contact configuration. All benchmarks were run on a PC with 64 GB of main memory and using one core of an Intel Xeon E5-1630 processor running at 3.7 GHz.

From the results we obtain, we observe that the complete computation time for one jump is inferior to the actual animation time. This is also true for our worst case scenario, the houses rooftops with Jumper-man. In this case, the large number of triangles that describe both the scene and the character explain the gap in the performances, due to the collision tests. As usually done in video games, using simplified meshes at the planning phase would probably allow us to obtain better performances without impacting the quality of the results. However we chose to preserve the complexity of the scene in this work, to demonstrate that the method scales well with the number of triangles.

More importantly, the critical observation is that both the trajectory query and the contact generation are real time in every scenario. We recall that these are the main contributions of our work, and the performances obtained allow us to consider interactive applications with more advanced animation methods.

6.7 Conclusions

This work introduces an efficient extension of the ballistic motion planner, able to compute and animate complex trajectories for jumping legged characters. In particular, contrary to previous works based on finite state machines or data-driven animation, the planner is able to automatically compute non-coplanar multi-contact configurations, without making hypotheses about the number of contacts required and their locations. As such, it is able to address arbitrary characters and environments.

Our planner is computationally competitive, thanks to the introduction of a low dimensional conservative criterion for verifying the non-slipping condition without explicitly computing the contacting limb configurations. This reduction of the problem dimension only approximates the kinematic constraints of the character, and in rare cases (less than 1% [Tonneau 2016b]), the wholebody contact generation at the animation phase will fail. However, in case of failure, the planner cannot determine if planning another path will be enough to solve the contact generation problem or if no solution exists. This limit is common to classic probabilistic planner, and is currently an open-ended question.

Regarding the quality of the obtained animations, we believe that the non-slipping criterion increases the plausibility of the obtained motion. We leave the validation of this hypothesis through a perception user study for future work. Besides, constraining the limb configurations to maintain the contacts during jump transitions reduces the lack of realism of the dynamic impulse contact model. Contrary to physics-based models, we do not require to explicitly compute the contact

force distribution for feeding a dynamic engine and obtain the animation. The limb accessible-space shapes $\mathbf{W}^k$ could be reduced to be more relevant for bending contacts. Such shapes would be learned from sampled limb configurations when limbs are in flexion, which is a notion that has to be firstly clarified.

Another current limitation of the method is that the parabola trajectory is followed by the geometrical center of the character instead of the COM. This can be improved by the introduction of a path-constraint after the wholebody animation, that projects any sampled configuration to a new one with its COM located on the parabola. The parabola also has to be followed with respect to a time parametrization, to visually express the character velocities and accelerations. This is crucial for the animation perception.

Another limit of the method is that limb motions and angular momentum are detached. Applying physics-based motions for limbs that are not in contact could increase the plausibility of the animation, as humanoids use their arms to counteract the gravity effects when they take off [Cheng 2008], and as the upper-body helps to stabilize the character when landing [Ashby 2006]. Mid-air re-orientation inspired by the falling cat problem [Kane 1969, Montgomery 1993] or more recent studies would improve realism during flight phases [Mather 2009, Bingham 2014, Zhao 2015, Shu 2016].

A current limitation of the model is that it does not consider the Euler equation in the planning phase. A feasibility study of the character's capacity to re-orientate itself knowing the contact locations could reject impracticable paths during the planning stage. A preliminary version of this study is proposed in Appendix C). This work would improve the plausibility of the ballistic trajectory because the rotation speeds produced with contact forces are limited by the physics.

Lastly, we aim at extending our planner to integrate other phases than jumping, to be able to alternate running or rolling sequences, for a larger spectrum of application. Our ballistic motion planner already inspired a recent work on legged multi-contact locomotion [Fernbach 2017]. The method is state-dependent and considers full centroidal dynamic constraints, contrary to our momentum decoupled approach.

Chapter 7

Conclusion

7.1 Contributions

Through this thesis, new applications benefiting from motion planning have arisen. Planning provides general tools to autonomously find a trajectory reaching a goal configuration and satisfying constraints. We based our studies on sampling-based planners as they are a breakthrough being probabilistically complete and avoiding local minima.

We have analyzed the planners limits in two different directions, corresponding to the thesis contributions:

1. How can classic path returned by these simple planners can be improved?

2. Can these planners be extended to new type of motions such as wholebody jumping?

First, a path-optimization tool based on a Linearly Constrained Quadratic Program has been provided. The method shortens the path length of a probabilistic planner output. The framework lies in a trade-off between the simplicity of blind random methods, and the complexity of heavy-computationally distance-based optimization techniques. A convergence study has been conducted to prove that the method cannot be stuck in an infinite loop. Simulations were conducted to show that the optimizer is time-competitive compared to random shortcuts. It even qualitatively surpasses them on high-DOF robots.

To address the second question, a ballistic motion planner was designed in two steps. First ballistic paths are planned for a point-mass and then for a simplified robot. Finally the trajectory is completed with contact generation and limb animation. The strength of the method is not to rely on an assumption on the legged character neither the environment nor the contact periodicity. Furthermore, planning with the simplified robot shape breaks the combinatorial complexity of contact generation. The planner parametrization is limited, with up to three main parameters (the friction coefficient and the velocity limitations), but they are critical for the problem completeness. Finally, the notion of constrained ballistic path is independent of the framework and can be re-used in another implementation, for instance in a jumping robot.

7.2 Limits

The path-optimizer parametrization can be automatized according to geometrical considerations. Computation time can be lowered if we want its time-competitiveness to be less questionable, in particular when the path has many waypoints. The method also lacks of flexibility in its constraint setting, as constraints may become too restrictive for the length reduction process. Constraint relaxation and re-creation could improve the optimization result. This is possible at the cost of introducing a new loop, which requires a tuned termination condition so that the algorithm is insured to converge.

The formulation of the ballistic motion planner contains several limits. Most of them are related to the simplifications made for computation efficiency to the detriment of solution existence. For instance, the impulse model of contact force constrains the takeoff and landing velocities, which is a harder condition than directly constraining the contact force. The two steps formulation of *first plan for a simplified shape and then generate the wholebody motions* is heuristic. Limb animation may fail because of a wrong planning or because the problem is infeasible. More striking, the quality result of animations is questionable from the computer graphics viewpoint. Relying on a few manually designed key-frames instead of motion capture appears less natural than classic animations. To remain independent of a database, physics-based motions obtained by constrained optimization or learning are also a good solution.

7.3 Perspectives

Figure 7.1: Snapshots[1] of a leopard executing multiple jumps. Contact changes from front to back legs during a jump transition are circled in red.

In the future, we plan to extend the ballistic motion planner and combine it with other motion sets, e.g. running and landing. There are also other ways to conceive jump, including contacts such as vaulting. Inspired of quadruped animals (see Figure 7.1), the possibility of changing contacts during a jump transition is also a challenge to tackle.

All these ideas can be addressed with new path types which will enrich the roadmap. At this stage, transitions between different types of motion do not have to be detailed. Once the path sequence is chosen, the motion generation will follow the chosen types of paths.

[1] Source: Rare Species Conservation Centre, Kent, UK. Video: https://youtu.be/2C3JEM8Szbw

To continue the angular momentum study, especially its generation through limb motions and trunk postures, we want to take inspiration from the falling cat problem (see Figure 7.2) and more recent work based on it [Kane 1969, Montgomery 1993, Mather 2009, Bingham 2014, Zhao 2015, Shu 2016].

Figure 7.2: A falling cat that lands on its feet despite its original orientation. © Ralph Cane - *Life* magazine

Appendix A

Appendix: Computation details of intersections

A.1 Intersection between a cone and a vertical plane

This section details how to analytically compute the intersection between a 3D cone $\mathcal{C}$ and a vertical plane π_θ.

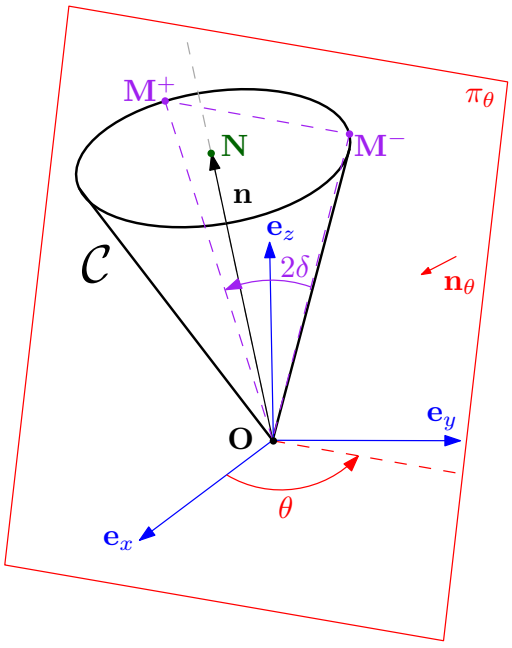


Figure A.1: Notations for the intersection between the cone $\mathcal{C}$ and the plane π_θ.

The notations are the followings:

- $\mathbf{O} = (0\ 0\ 0)^T$ is the cone $\mathcal{C}$ apex.
- $\mathbf{n} = (n_x\ n_y\ n_z)^T$ is the cone $\mathcal{C}$ normed direction.
- μ is the tangent friction coefficient. It is equal to $\tan(\phi)$ with ϕ the half-apex angle of $\mathcal{C}$. We pose $\mu_{12} = 1 + \mu^2$.
- $\mathcal{P}$ is the plane passing by $\mathbf{N}$ and of normal $\mathbf{n}$. As $||\mathbf{n}|| = 1$, $\mathcal{P}$ verifies:

$$n_x\, x + n_y\, y + n_z\, z = 1 \tag{A.1}$$

- Let $\mathcal{S}$ be the sphere of center $\mathbf{N} = (n_x\ n_y\ n_z)^T$ and of radius μ. $\mathcal{S}$ equation is the following:
$$(x-n_x)^2+(y-n_y)^2+(z-n_z)^2=\mu^2 \tag{A.2}$$
The circle $\mathscr{C}$ of center $\mathbf{N}$, radius μ and normal $\mathbf{n}$ is defined as the intersection of $\mathcal{S}$ and $\mathcal{P}$. By definition, $\mathscr{C}$ belongs to the cone surface.

  By combining Equations (A.1-A.2), $\mathscr{C}$ equations can be simplified as it follows:
$$\begin{aligned} x^2+y^2+z^2&=\mu_{12}\\ n_x\,x+n_y\,y+n_z\,z&=1 \end{aligned} \tag{A.3}$$
Note that $\mathscr{C}$ is also the intersection between $\mathcal{S}'$ and $\mathcal{P}$, where $\mathcal{S}'$ is the sphere of center $\mathbf{O}$ and radius μ_{12}.

- π_θ is the plane passing by $\mathbf{O}$ and of normal $\mathbf{n}_\theta = (\sin(\theta)\ -\cos(\theta)\ 0)^T$.
So π_θ equation is:
$$x\sin(\theta)-y\cos(\theta)=0 \tag{A.4}$$

- For convenience: $\cos(\theta)=c\theta$, $\sin(\theta)=s\theta$, $\tan(\theta)=t\theta$ (if defined).

Given that π_θ goes through $\mathbf{O}$, π_θ intersects the cone $\mathcal{C}$ (with a result different from $\mathbf{O}$) if and only if the plane π_θ intersects the circle $\mathscr{C}$. We denote by $\mathbf{M}^+ = (x^+_{\mathbf{M}}\ y^+_{\mathbf{M}}\ z^+_{\mathbf{M}})^T$ and $\mathbf{M}^- = (x^-_{\mathbf{M}}\ y^-_{\mathbf{M}}\ z^-_{\mathbf{M}})^T$ the resulting points of the intersection between π_θ and $\mathscr{C}$. Note that they may be equal, and also be equal to $\mathbf{O}$. From Equations (A.3-A.4), the intersection computation is analytically detailed within the cases of Table A.1.

Finally, when the intersection does not result in $\mathbf{O}$, $(\mathbf{O}, \mathbf{OM}^+, \mathbf{OM}^-)$ forms a 2D cone of apex $\mathbf{O}$ and included in π_θ. Its half-apex angle δ is given by:
$$\delta = \frac{1}{2}\,\mathrm{atan2}(||\mathbf{OM}^+ \times \mathbf{OM}^-||, \mathbf{OM}^+ \cdot \mathbf{OM}^-) \tag{A.5}$$

Cases	**Conditions**	**$(\mathbf{M}^+, \mathbf{M}^-)$ coordinates**
I	$n_z \neq 0$ $\theta \neq \pm\frac{\pi}{2}$	$A = 1 + t\theta^2 + (n_x + t\theta n_y)^2/n_z^2$ $B = -2(n_x + t\theta n_y)/n_z^2$ $C = 1/n_z^2 - \mu_{12}$
I.a	$n_z \neq 0$ $\theta \neq \pm\frac{\pi}{2}$ $B^2 - 4AC \geq 0$	$x_{\mathbf{M}}^{\pm} = 0.5(-B \pm \sqrt{B^2 - 4AC})/A$ $y_{\mathbf{M}}^{\pm} = t\theta x_{\mathbf{M}}^{\pm}$ $z_{\mathbf{M}}^{\pm} = (1 - n_x x_{\mathbf{M}}^{\pm} - n_y y_{\mathbf{M}}^{\pm})/n_z$
I.b	$n_z \neq 0$ $\theta \neq \pm\frac{\pi}{2}$ $B^2 - 4AC < 0$	$\mathbf{M}^{\pm} = \mathbf{O}$
II	$n_z \neq 0$ $\theta = \pm\frac{\pi}{2}$	$A = 1 + (n_y/n_z)^2$ $B = -2n_y/n_z^2$ $C = 1/n_z^2 - \mu_{12}$
II.a	$n_z \neq 0$ $\theta = \pm\frac{\pi}{2}$ $B^2 - 4AC \geq 0$	$x_{\mathbf{M}}^{\pm} = 0$ $y_{\mathbf{M}}^{\pm} = 0.5(-B \pm \sqrt{B^2 - 4AC})/A$ $z_{\mathbf{M}}^{\pm} = (1 - n_y y_{\mathbf{M}}^{\pm})/n_z$
II.b	$n_z \neq 0$ $\theta = \pm\frac{\pi}{2}$ $B^2 - 4AC < 0$	$\mathbf{M}^{\pm} = \mathbf{O}$
III	$n_z = 0$ $\theta \neq \pm\frac{\pi}{2}$ $n_y \neq 0$ $n_x + t\theta n_y \neq 0$	$x_{\mathbf{M}}^{\pm} = 1/(n_x + t\theta n_y)$ $y_{\mathbf{M}}^{\pm} = (1 - n_x x_{\mathbf{M}}^{\pm})/n_y$ $z_{\mathbf{M}}^{\pm} = \pm\sqrt{\mu_{12} - (1 + t\theta^2) x_{\mathbf{M}}^{\pm 2}}$
IV	$n_z = 0$ $\theta \neq \pm\frac{\pi}{2}$ $n_y \neq 0$ $n_x + t\theta n_y = 0$	$\mathbf{M}^{\pm} = \mathbf{O}$
V	$n_z = 0$ $\theta \neq \pm\frac{\pi}{2}$ $n_y = 0$	$x_{\mathbf{M}}^{\pm} = 1$ $y_{\mathbf{M}}^{\pm} = t\theta$ $z_{\mathbf{M}}^{\pm} = \pm\sqrt{\mu_{12} - (1 + t\theta^2)}$
VI	$n_z = 0$ $\theta = \pm\frac{\pi}{2}$ $n_y \neq 0$	$x_{\mathbf{M}}^{\pm} = 0$ $y_{\mathbf{M}}^{\pm} = 1/n_y$ $z_{\mathbf{M}}^{\pm} = \pm\sqrt{\mu_{12} - 1/n_y^2}$
VII	$n_z = 0$ $\theta = \pm\frac{\pi}{2}$ $n_y = 0$	$\mathbf{M}^{\pm} = \mathbf{O}$

Table A.1: Case-study of the different results of the plane-circle intersection.

A.2 Intersection between a convex sum of cones and a vertical plane

We consider multiple contact cones $\mathcal{K}_k$, $1 \geq k \geq n$, assuming that there is no force-closure. We denote the convex cone $\mathcal{K}$ resulting from the Minkowski sum of multiple cones. This section details how to analytically compute the intersection between $\mathcal{K}$ and a vertical plane π_θ. If only one contact one is considered, we refer the reader to the previous section for the intersection computation.
We consider the following notations:

- $n \geq 2$ is the number of contact cones considered for the Minkowski sum.
- All the cone apexes are equal to $\mathbf{O} = (0\ 0\ 0)^T$.
- All the cone friction coefficients are equal to μ which is equal to $\tan(\phi)$ with ϕ the half-apex angles. We pose $\mu_{12} = 1 + \mu^2$.
- $\mathbf{n}_i = (n_{ix}\ n_{iy}\ n_{iz})^T$ is the cone $\mathcal{K}_i$ direction.
- We denote by $\mathcal{Z}_{ij}$ the convex sum of two cones $\mathcal{K}_i$ and $\mathcal{K}_i$ (see Figure A.2).
- π_θ is the plane passing by $\mathbf{O}$ and of normal $\mathbf{n}_\theta = (\sin(\theta)\ -\cos(\theta)\ 0)^T$.

  So π_θ equation is:
  $$x\sin(\theta) - y\cos(\theta) = 0 \tag{A.6}$$
- For convenience: $\cos(\theta) = c\theta$, $\sin(\theta) = s\theta$, $\tan(\theta) = t\theta$ (if defined).

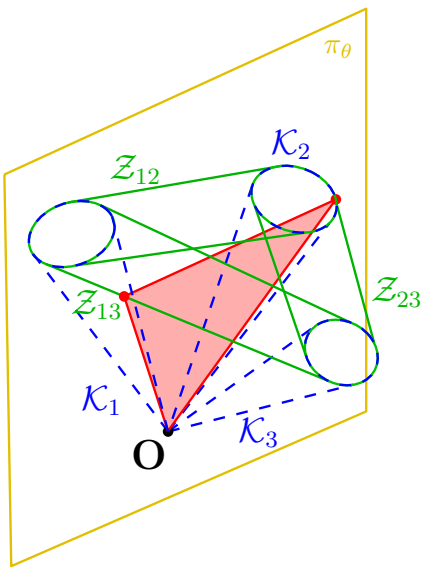


Figure A.2: Illustration of convex sums $\mathcal{Z}_{ij}$ (green) in the case $n = 3$. An example of intersection with π_θ is given (red) to show that it also intersects $\mathcal{Z}_{ij}$ shapes.

The analytical formulation of $\mathcal{K}$ does not exist. However, getting its intersection with π_θ is possible. As convex sums $\mathcal{Z}_{ij}$ constitute the borders of $\mathcal{K}$, the plane π_θ intersects the convex-cone $\mathcal{K}$ if and only if π_θ intersects at least one convex sum $\mathcal{Z}_{ij}$. Note that for all $\mathcal{K}_i$ and $\mathcal{K}_j$ cones, the intersection between $\mathcal{K}$ and π_θ is always included in the convex union of the intersections between $\mathcal{Z}_{ij}$ and π_θ.

When the intersection between $\mathcal{K}$ and π_θ is not reduced to $\mathbf{O}$, we denote it as a 2D convex-cone $\mathcal{K}_\mathbf{c}$.

Property 5.

$$\mathcal{K}_\mathbf{c} = \{\mathbf{F} \in \pi_\theta \mid \exists (\mathbf{f}_i)_{i\in[1..N_f]} \in (\mathcal{C}_k)_{k\in[1..n]}, \mathbf{F} = \sum_i^{N_f} \mathbf{f}_i\}$$

All vectors in the $\mathcal{K}_\mathbf{c}$ can be written as a resultant of forces which belong to contact cones $\mathcal{K}_k$. All resultant of forces belonging to contact cones, that also belongs to π_θ, is included in $\mathcal{K}_\mathbf{c}$.

Proof. By definition, the $\mathcal{K}_\mathbf{c}$ represents the intersection between π_θ and $\mathcal{K}$.
$\Rightarrow$ All vector that belongs to a subset of $\mathcal{K}$, belongs to $\mathcal{K}$, so can be written as a combination of forces belonging to the contact cones.
$\Leftarrow$ Let us assume that their exist a resultant $\mathbf{F}$ of contact-forces that belongs to π_θ. $\mathbf{F}$ being a resultant of forces that lie in contact cones, $\mathbf{F}$ belongs to $\mathcal{K}$. As $\mathbf{F}$ also belongs to π_θ, $\mathbf{F}$ lies in the intersection of $\mathcal{K}$ and π_θ, i.e. $\mathcal{K}_\mathbf{c}$. So $\mathbf{F}$ belongs to $\mathcal{K}_\mathbf{c}$. □

Thus, computing $\mathcal{K}_\mathbf{c}$ results in computing all the intersections with the sub-convex-cones $\mathcal{Z}_{ij}$ and then, computing their convex union. The framework is given in Algorithm 10.

Algorithm 10 Process to compute $\mathcal{K}_\mathbf{c}$ from contact cones $\mathcal{K}_k$ and plane π_θ.

Input: n cones $\mathcal{K}_k$, π_θ
Output: Intersection points $\mathbf{M}^+$ and $\mathbf{M}^-$, half-apex angle φ^{cc}

```
for 1 ≤ i ≤ n do
    for i < j ≤ n do
        M_list ← conePlaneIntersection(K_i, π_θ)
        if (isNotParallel(Z_ij, π_θ)) then
            Q_list ← convexConePlaneInters(Z_ij, π_θ)
[M+, M−, φ^Kc] ← computeMaxRange(Q_list, π_θ)
return [M+, M−, φ^Kc]
```

For convenience, we detail the process in the case $n = 2$ ($i = 1$ and $j = 2$).

We precise the following notations:

- $\mathbf{n}_1$ notation is simplified to $(n_z\ n_y\ n_z)^T$, $\mathbf{n}_2 = (n_{2x}\ n_{2y}\ n_{2z})^T$
- $\mathbf{N} = (n_x\ n_y\ n_z)^T$.

- $\mathbf{t}_{12} = (t_x\ t_y\ t_z)^T$ is the inter-cones direction from $\mathcal{K}_2$ to $\mathcal{K}_1$: $\mathbf{t}_{12} = \mathbf{n}_1 - \mathbf{n}_2$.
- $C_{yz} = n_y n_{2z} - n_z n_{2y}$; $C_{zx} = n_z n_{2x} - n_x n_{2z}$; $C_{xy} = n_x n_{2y} - n_y n_{2x}$
- $n_{12} = \mathbf{n}_1 \cdot \mathbf{n}_2$; $n_{12z} = n_z n_{12} - n_{2z}$
- $\mathbf{P} = (x_\mathbf{P}\ y_\mathbf{P}\ z_\mathbf{P})^T$ is a point of the cone $\mathcal{K}_1$.
- $\mathbf{Q} = (x_\mathbf{Q}\ y_\mathbf{Q}\ z_\mathbf{Q})^T$ is a point of the plane π_θ.

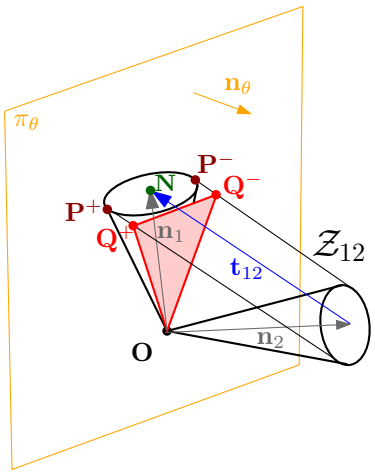


Figure A.3: Notations for the convex sum $\mathcal{Z}_{12}$ of two cones and its intersection with the plane π_θ.

We detail the functions of Algorithm 10:

- isNotParallel: Returns $false$ if π_θ is parallel to $\mathbf{t}_{12}$, i.e. $\mathbf{n}_\theta \cdot \mathbf{t}_{12} = 0$. Thus the parallelism condition is equivalent to:

$$s\theta\, t_x - c\theta\, t_y = 0$$

- conePlaneIntersection$(\mathcal{K}_i, \pi_\theta)$ computes the points of the intersection between $\mathcal{K}_i$ and π_θ. The function follows the method of the previous section. Resulting points $(\mathbf{M}^+, \mathbf{M}^-)$ are added to a list of points $\mathbf{Q}_{list}$.

- convexConePlaneIntersection$(\mathcal{Z}_{ij}, \pi_\theta)$: computes the part of the points of the intersection between $\mathcal{Z}_{ij}$ and π_θ. This intersection does not include points of the cones $\mathcal{K}_i$ and $\mathcal{K}_j$ as this is addressed by another method. Resulting points $(\mathbf{Q}^+, \mathbf{Q}^-)$ are added to the list of points $\mathbf{Q}_{list}$.

  Note that this function is only used if the parallelism condition is not verified. Otherwise, as we look for the convex union of the resulting points, points from

conePlaneIntersection include points of the intersection between $\mathcal{Z}_{ij}$ and π_θ.

We construct the intersection from a geometrical approach (see Figure A.3). We first compute extremities of $\mathcal{Z}_{12}$ on the cone $\mathcal{K}_1$, denoted as $(\mathbf{P}^+, \mathbf{P}^-)$ points. Then we project them on π_θ following the direction $\mathbf{t}_{12}$ to obtain $(\mathbf{Q}^+, \mathbf{Q}^-)$ points.

$\mathbf{P}$ coordinates verify the following equations:

- $\mathbf{NP} \cdot \mathbf{n} = 0$, therefore:

$$x_\mathbf{P}\, n_x + y_\mathbf{P}\, n_y + z_\mathbf{P}\, n_z = 1 \tag{A.7}$$

- $\mathbf{P} \in \mathscr{C}$ where $\mathscr{C}$ is the circle of center $\mathbf{N} = (n_z\ n_y\ n_z)^T$, radius μ and in the plane of normal $\mathbf{n}$ (thus $\mathscr{C}$ belongs to the cone surface). Thus $\mathbf{P}$ verifies:

$$x_\mathbf{P}^2 + y_\mathbf{P}^2 + z_\mathbf{P}^2 = \mu_{12} \tag{A.8}$$

- $\mathbf{NP} \cdot \mathbf{t}_{12} = 0$, therefore:

$$x_\mathbf{P}\, n_{2x} + y_\mathbf{P}\, n_{2y} + z_\mathbf{P}\, n_{2z} = n_{12} \tag{A.9}$$

When solving Equations (A.6-A.7-A.8-A.9), we obtain two solutions $(\mathbf{P}^+, \mathbf{P}^-)$ depending on the cases exposed in the Table A.2.

The projection of $\mathbf{P}$ on π_θ with the direction $\mathbf{t}_{12}$ is described by the following equations:

- $\mathbf{Q} \in \pi_\theta$, (note that $\mathbf{O} \in \pi_\theta$) therefore:

$$x_\mathbf{Q} \sin(\theta) - y_\mathbf{Q} \cos(\theta) = 0 \tag{A.10}$$

- $\mathbf{PQ} \times \mathbf{t}_{12} = 0$, thus:

$$\exists\ \alpha \in \mathbb{R} \mid \mathbf{PQ} = \alpha\, \mathbf{t}_{12} \tag{A.11}$$

When solving Equations (A.10) and (A.11), we obtain two solutions $(\mathbf{Q}^+, \mathbf{Q}^-)$ depending on the cases exposed in the Table A.3. Then, $(\mathbf{Q}^+, \mathbf{Q}^-)$ are added to the list of points $\mathbf{Q}_{list}$.

- computeMaxRange: computes the two vectors delimiting the maximal 2D angular sector $\varphi^{\mathcal{K}_\mathbf{c}}$ from all the vectors built with $\mathbf{Q}_{list}$ (see Figure A.4). ψ_i is the angle $\angle(\mathbf{e}_{x_\theta}, \mathbf{OQ}^i_{list})$:

$$\psi_i = \text{atan2}(z_i, x_{\theta i})$$

Cases	Conditions	$(\mathbf{P}^+, \mathbf{P}^-)$ coordinates
I	$n_z \neq 0$	$A = 1 + C_{zx}^2/C_{yz}^2 + (n_y C_{zx} + n_x C_{yz})^2/(n_z C_{yz})^2$
		$B = -2C_{zx} n_{12z}/C_{yz}^2$
		$\quad -2(n_y C_{zx} + n_x C_{yz})(C_{yz} + n_y n_{12z})/(n_z C_{yz})^2$
		$C = n_{12z}^2/C_{yz}^2 + (C_{yz} + n_y n_{12z})^2/(n_z C_{yz})^2 - \mu_{12}$
	$C_{yz} \neq 0$	$x_{\mathbf{P}}^{\pm} = 0.5(-B \pm \sqrt{B^2 - 4AC})/A$
		$y_{\mathbf{P}}^{\pm} = C_{zx} x_{\mathbf{P}}^{\pm}/C_{yz} - n_{12z}/C_{yz}$
		$z_{\mathbf{P}}^{\pm} = (1 - n_x x_{\mathbf{P}}^{\pm} - n_y y_{\mathbf{P}}^{\pm})/n_z$
II	$n_z \neq 0$	$x_{\mathbf{P}}^{\pm} = n_{12z}/C_{zx}$
		$A = 1 + n_y^2/n_z^2$
		$B = -2n_y(1 - n_x x_{\mathbf{P}}^{\pm})/n_z^2$
		$C = (1 - n_x x_{\mathbf{P}}^{\pm})^2/n_z^2 + x_{\mathbf{P}}^{\pm 2} - \mu_{12}$
	$C_{yz} = 0$	$y_{\mathbf{P}}^{\pm} = 0.5(-B \pm \sqrt{B^2 - 4AC})/A$
		$z_{\mathbf{P}}^{\pm} = (1 - n_x x_{\mathbf{P}}^{\pm} - n_y y_{\mathbf{P}}^{\pm})/n_z$
III	$n_z = 0$	$A = 1 + n_x^2/n_y^2 + C_{xy}^2/(n_y n_{2z})^2$
		$B = -2n_x/n_y^2 + 2C_{xy}(n_y n_{12} - n_{2y})/(n_y n_{2z})^2$
	$n_y \neq 0$	$C = 1/n_y^2 + (n_y n_{12} - n_{2y})^2/(n_y n_{2z})^2 - \mu_{12}$
		$x_{\mathbf{P}}^{\pm} = 0.5(-B \pm \sqrt{B^2 - 4AC})/A$
	$n_{2z} \neq 0$	$y_{\mathbf{P}}^{\pm} = (1 - n_x x_{\mathbf{P}}^{\pm})/n_y$
		$z_{\mathbf{P}}^{\pm} = (C_{xy} x_{\mathbf{P}}^{\pm} + n_y n_{12} - n_{2y})/(n_y n_{2z})$
IV	$n_z = 0$	$x_{\mathbf{P}}^{\pm} = (n_{2y} - n_y n_{12})/C_{xy}$
	$n_y \neq 0$	$y_{\mathbf{P}}^{\pm} = (1 - n_x x_{\mathbf{P}}^{\pm})/n_y$
	$n_{2z} = 0$	$z_{\mathbf{P}}^{\pm} = \pm\sqrt{\mu_{12} - x_{\mathbf{P}}^{\pm 2} - y_{\mathbf{P}}^{\pm 2}}$
	$C_{xy} \neq 0$	
V	$n_z = 0$	$x_{\mathbf{P}}^{\pm} = n_x$
	$n_y \neq 0$	$y_{\mathbf{P}}^{\pm} = (1 - n_x x_{\mathbf{P}}^{\pm})/n_y$
	$n_{2z} = 0$	$z_{\mathbf{P}}^{\pm} = \pm\sqrt{\mu_{12} - x_{\mathbf{P}}^{\pm 2} - y_{\mathbf{P}}^{\pm 2}}$
	$C_{xy} = 0$	
VI	$n_z = 0$	$x_{\mathbf{P}}^{\pm} = 1$
	$n_y = 0$	$y_{\mathbf{P}}^{\pm} = \pm\mu\sqrt{n_{2z}^2/(n_{2z}^2 + n_{2y}^2)}$
	$n_{2z} \neq 0$	$z_{\mathbf{P}}^{\pm} = -n_{2y} y_{\mathbf{P}}^{\pm}/n_{2z}$
VII	$n_z = 0$	$x_{\mathbf{P}}^{\pm} = 1$
	$n_y = 0$	$y_{\mathbf{P}}^{\pm} = 0$
	$n_{2z} = 0$	$z_{\mathbf{P}}^{\pm} = \pm\mu$

Table A.2: Case-study of the different results of $\mathcal{Z}_{12}$ extreme points $(\mathbf{P}^+, \mathbf{P}^-)$ located on $\mathcal{K}_1$. Results are valid if $\mathbf{n} \times \mathbf{n}_2 \neq \mathbf{O}$. Note that the case V is a force-closure case.

Cases	Conditions	$(\mathbf{Q}^+, \mathbf{Q}^-)$ coordinates
I	$\theta \neq \pm\frac{\pi}{2}$	$\alpha = (t\theta x_{\mathbf{P}}^{\pm} - y_{\mathbf{P}}^{\pm})/(t_y - t_x t\theta)$ $x_{\mathbf{Q}}^{\pm} = \alpha t_x + x_{\mathbf{P}}^{\pm}$ $y_{\mathbf{Q}}^{\pm} = \alpha t_y + y_{\mathbf{P}}^{\pm}$ $z_{\mathbf{Q}}^{\pm} = \alpha t_z + z_{\mathbf{P}}^{\pm}$
II	$\theta = \pm\frac{\pi}{2}$	$\alpha = -x_{\mathbf{P}}^{\pm}/tx$ $x_{\mathbf{Q}}^{\pm} = 0$ $y_{\mathbf{Q}}^{\pm} = \alpha t_y + y_{\mathbf{P}}^{\pm}$ $z_{\mathbf{Q}}^{\pm} = \alpha t_z + z_{\mathbf{P}}^{\pm}$

Table A.3: Case-study of the different results $(\mathbf{Q}^+, \mathbf{Q}^-)$ obtained from the projection of extreme points $(\mathbf{P}^+, \mathbf{P}^-)$ on π_θ. Results are valid if $\mathbf{n}_\theta \cdot \mathbf{t}_{12} \neq 0$.

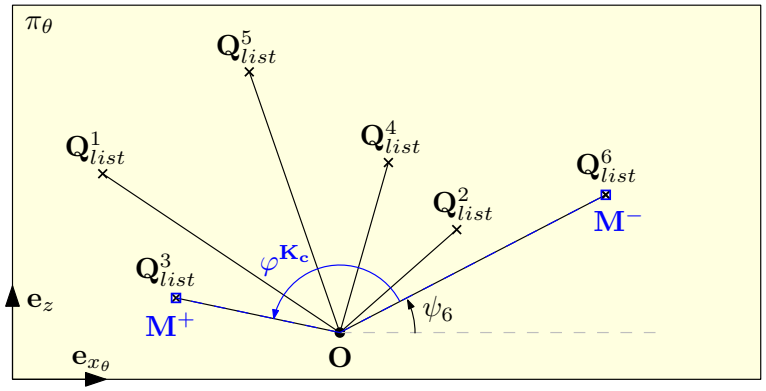


Figure A.4: Example of a convex cone $\mathcal{K}_\mathbf{c}$ delimited by $(\mathbf{O}, \mathbf{OM}^+, \mathbf{OM}^-)$ and of half-apex angle $\varphi^{\mathcal{K}_\mathbf{c}}$.

Border points are given by:

$$\mathbf{M}^+ = \underset{\mathbf{Q}^i_{list}}{\arg\min}(\psi_i) \quad \text{and} \quad \mathbf{M}^- = \underset{\mathbf{Q}^i_{list}}{\arg\max}(\psi_i)$$

The half-apex angle of the 2D convex cone $\mathcal{K}_\mathbf{c}$ becomes:

$$\varphi^{\mathcal{K}_\mathbf{c}} = \frac{1}{2}\,\text{atan2}(||\mathbf{OM}^+ \times \mathbf{OM}^-||, \mathbf{OM}^+ \cdot \mathbf{OM}^-)$$

Appendix B

Appendix: Rotation effect

In this section we explain the computation details to re-orientate a character so that it intuitively lays on its limbs on the surface and it faces the direction given by an angle.

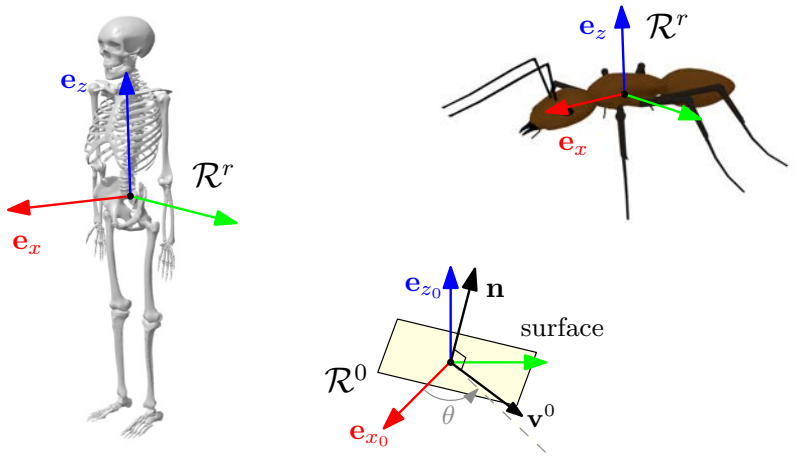


Figure B.1: Local frame definitions for a humanoid (left) and a non-humanoid character (right). $\mathbf{v}^0$ is the desired facing direction, given a surface normal $\mathbf{n}$ and an orientation angle θ.

A is the unknown rotation matrix describing the character orientation from its local frame $\mathcal{R}^r$ of superscript r, to the global frame $\mathcal{R}^0$ of superscript 0. We consider a surface of normal denoted $\mathbf{n}$ and a perpendicular direction $\mathbf{n}_\theta$ to the desired θ angle direction, so that $\mathbf{n}_\theta = (\sin(\theta) \;\; -\cos(\theta) \;\; 0)^T$. According to the Figure B.1 definitions, the character standing direction is given by $\mathbf{e}_z$ and its facing direction is $\mathbf{e}_x$. By definition, $\mathbf{e}_z^r = (0 \;\; 0 \;\; 1)^T$ and $\mathbf{e}_x^r = (1 \;\; 0 \;\; 0)^T$. So the rotation matrix becomes:

$$A = (\mathbf{e}_x^0 \;\; \mathbf{e}_z^0 \times \mathbf{e}_x^0 \;\; \mathbf{e}_z^0) \in \mathbb{R}^{3\times 3}$$

The identification of these directions in $\mathcal{R}^0$ depends on the desired orientation for the character. We consider two cases:

1. **The standing case**: this case includes all characters that are not humanoid. Humanoids are considered if the surface is almost horizontal, which is tested by the arbitrary condition:

   $$\mathbf{n} \,.\, \mathbf{e}_z^0 > 0.707$$

   This assumption is made because it is unusual to see a humanoid character

jumping only with its legs on a vertical surface, we rather think that it will also use its arms. By identification, $\mathbf{e}_z^0 = \mathbf{n}$ and $\mathbf{e}_x^0$ verifies the following conditions:

$$\begin{cases} ||\mathbf{e}_x^0|| = 1 \\ \mathbf{e}_x^0 \perp \mathbf{n} \\ \mathbf{e}_x^0 . \mathbf{n}_\theta = 0 \end{cases}$$

2. **The on all fours case**: this case is complementary to the previous one and only concerns the humanoids. The method is similar, simply switching $\mathbf{e}_z^0$ and $-\mathbf{e}_x^0$. So $\mathbf{e}_x^0 = -\mathbf{n}$ and $\mathbf{e}_z^0$ verifies:

$$\begin{cases} ||\mathbf{e}_z^0|| = 1 \\ \mathbf{e}_z^0 \perp \mathbf{n} \\ \mathbf{e}_z^0 . \mathbf{n}_\theta = 0 \end{cases}$$

The system solution is identical in both cases. We detail the solving for an arbitrary direction vector $\mathbf{v^0} = (a\ b\ c)^T$. Thus the previous conditions become:

$$\begin{cases} a^2 + b^2 + c^2 = 1 \\ a\,n_x + b\,n_y + c\,n_z = 0 \\ a\sin(\theta) - b\cos(\theta) = 0 \end{cases}$$

We split our resolution into the different cases presented in Table B.1.

Cases	**Conditions**	$\mathbf{v^0} = (a\ b\ c)^T$ **coordinates**
Definition	$\theta \in]-\frac{\pi}{2};\frac{\pi}{2}]$	$S = 1$
Definition	$\theta \notin]-\frac{\pi}{2};\frac{\pi}{2}]$	$S = -1$
I	$\theta \neq \pm\frac{\pi}{2}$ $n_z \neq 0$	$a = S\sqrt{n_z^2/(n_z^2 + n_z^2 t\theta^2 + (n_x + n_y t\theta)^2)}$ $b = t\theta\, a$ $c = -a\,(n_x + n_y t\theta)/n_z$
II	$\theta \neq \pm\frac{\pi}{2}$ $n_z = 0$	$\mathbf{v^0} = \mathbf{e}_z^0$
III	$\theta = \pm\frac{\pi}{2}$ $n_y \neq 0$	$a = 0$ $b = -n_z\, c/n_y$ $c = S\sqrt{n_y^2/(n_y^2 + n_z^2)}$
IV	$\theta = \pm\frac{\pi}{2}$ $n_y = 0$ $n_x = 1$	$a = 0$ $b \in \mathbb{R}$ $c \in \mathbb{R}$
V	$\theta = \pm\frac{\pi}{2}$ $n_y = 0$ $n_x \neq 1$	$\mathbf{v^0} = \mathbf{e}_y^0$

Table B.1: Case-study of the different results for $\mathbf{v^0}$. $t\theta$ stands for $\tan(\theta)$ when defined.

Appendix C

Preliminary work: angular momentum feasibility study

Let us consider a parabola sequence with orientations $\bar{\mathbf{s}}$ determined by the function rotateAlongPath in Algorithm 8. We iteratively determine if the orientation change along each jump is feasible given the previous jump angular momentum, the contact locations and the character dynamic properties.

We remind the Newton-Euler equations of dynamics that apply during a contact phase:

$$m\ddot{\mathbf{c}} = \sum_{i=1}^{k} \mathbf{f}_i + m\mathbf{g} \qquad \dot{\mathbf{L}} = \sum_{i=1}^{k} (\mathbf{P}_i - \mathbf{c}) \times \mathbf{f}_i$$

where k is the number of contacts with the environment, $\mathbf{P}_i$ are the contact point positions, $\mathbf{f}_i$ the contact impulse-forces and $\dot{\mathbf{L}}$ the angular momentum variation.

The second equation can be reformulated as:

$$m\mathbf{c} \times (\ddot{\mathbf{c}} - \mathbf{g}) + \dot{\mathbf{L}} = \sum_{i=1}^{k} (\mathbf{P}_i \times \mathbf{f}_i)$$

In order to compute a COM acceleration and contact forces that satisfy the equations of dynamics, the non-sliding constraints and the velocity limitations, one possibility is to extend the centroidal cone to a 6 dimensional wrench-cone, following the formulation of [Caron 2015]. However, this method may change the resultant of the contact forces, and so may not be valid regarding the landing and takeoff velocities of the contact phase, that were found during the ballistic planning process. Instead, we chose to limit the problem to a feasibility study.

We introduce the following notations:

- Given two consecutive parabolas of superscripts j and $j+1$ (the process is the same for one parabola at the extremity of the path, adjusting the values that become null),
- Δt is the estimated duration of contact between the two jumps,
- T is the duration of flight along a parabola, $T = \frac{X_\theta}{\dot{x}_{\theta_s}}$,
- $I_{\mathbf{c}}$ is the character momentum of inertia expressed at its COM [Orin 2013]. For simplification, $I_{\mathbf{c}}$ is assumed to be constant and it is computed by assimilating all characters to cylinders, the COM being located at the center.

The following matrices are the momenta of inertia for cylinders of axis $\mathbf{e}_z$ and $\mathbf{e}_x$ respectively:

$$I_{\mathbf{e}_z} = \begin{pmatrix} \frac{1}{12}m(3r^2+h^2) & 0 & 0 \\ 0 & \frac{1}{12}m(3r^2+h^2) & 0 \\ 0 & 0 & \frac{1}{2}mr^2 \end{pmatrix}$$

$$I_{\mathbf{e}_x} = \begin{pmatrix} \frac{1}{2}mr^2 & 0 & 0 \\ 0 & \frac{1}{12}m(3r^2+h^2) & 0 \\ 0 & 0 & \frac{1}{12}m(3r^2+h^2) \end{pmatrix}$$

where m is the cylinder mass, R is its radius and H its height.

Character	m (kg)	R (m)	H (m)
Ant	25	0.1	0.55
Skeleton	20	0.23	1.44
Jumper-man	70	0.245	1.61
Lamp	70	0.099	0.2
Kangaroo	90	0.23	0.9
Frog	20	0.09	0.25

Table C.1: Character parameters for momentum of inertia computation.

The angular momentum during a flight is given by:

$$\mathbf{L} = I_{\mathbf{c}} \frac{\Delta\boldsymbol{\Theta}}{T}$$

where $\Delta\boldsymbol{\Theta}$ is the difference of the character orientations at takeoff and landing.

The feasibility problem is defined as trying to find contact forces f_i under the following constraints:

$$\mathbf{F_c} = \frac{\dot{\mathbf{c}}_l - \dot{\mathbf{c}}_t}{m\Delta t} = \sum_{i=1}^{k} \mathbf{f}_i$$

$$\dot{\mathbf{L}} = \frac{\mathbf{L}^{j+1} - \mathbf{L}^j}{\Delta t} = \sum_{i=1}^{k} (\mathbf{P}_i \times \mathbf{f}_i) - m\mathbf{c} \times (\ddot{\mathbf{c}} - \mathbf{g})$$

This Linear Problem can be solved numerically. If a solution is likely to be found, the angular momentum variation, and so the re-orientation, are feasible, and so the oriented jump should be plausible. Otherwise, the character re-orientation will appear as unnatural.

This feasibility study can be implemented in a function that would take action after rotateAlongPath in Algorithm 8. If the feasibility is proved, the initial sequence $\bar{\mathbf{s}}$ is conserved. Otherwise, the related parabolas are invalidated in the roadmap. Then, the motion planner is started again to replace the removed paths with new ones.

Bibliography

[Absil 2008] P. A. Absil, R. Mahony and R. Sepulchre. Optimization algorithms on matrix manifolds. Princeton University Press, Princeton (NJ), 2008. (Cited in page 11.)

[Amato 1996] N. Amato and Y. Wu. *A randomized roadmap method for path and manipulation planning*. In IEEE International Conference on Robotics and Automation (ICRA), volume 1, pages 113–120 vol.1, 1996. (Cited in page 53.)

[Arikan 2003] O. Arikan, D.A. Forsyth and J.F. O'Brien. *Motion Synthesis from Annotations*. ACM Trans. Graph., vol. 22, no. 3, pages 402–408, 2003. (Cited in page 40.)

[Ashby 2006] B.M. Ashby and S.L. Delp. *Optimal control simulations reveal mechanisms by which arm movement improves standing long jump performance*. Journal of Biomechanics, vol. 39, no. 9, pages 1726 – 1734, 2006. (Cited in page 80.)

[Barraquand 1991] J. Barraquand and J.-C. Latombe. *Robot Motion Planning: A Distributed Representation Approach*. International Journal of Robotics Research (IJRR), vol. 10, no. 6, pages 628–649, 1991. (Cited in page 6.)

[Batts 2017] Z. Batts, J. Kim and K. Yamane. Untethered one-legged hopping in 3d using linear elastic actuator in parallel (leap), pages 103–112. Springer International Publishing, Cham, 2017. (Cited in page 38.)

[Betts 2009] J.T. Betts. Practical methods for optimal control and estimation using nonlinear programming. Cambridge University Press, New York (NY), 2nd ed., 2009. (Cited in page 7.)

[Bingham 2014] J.T. Bingham, R. Lee, R.N. Haksar, J. Ueda and C.K. Liu. *Orienting in Mid-air through Configuration Changes to Achieve a Rolling Landing for Reducing Impact after a Fall*. In IEEE International Conference on Intelligent Robots and Systems (IROS), Chicago, Illinois, 2014. (Cited in pages 80 and 83.)

[Boston Dynamics 2012] Boston Dynamics. *Description of the Sand Flea robot*, 2012. https://www.bostondynamics.com/sandflea. (Cited in page 38.)

[Boston Dynamics 2017] Boston Dynamics. *Description of the Handle robot*, 2017. https://www.bostondynamics.com/handle. (Cited in page 38.)

[Boyd 2004] S. Boyd and L. Vandenberghe. Convex optimization. Cambridge University Press, 2004. (Cited in page 63.)

[Brady 1983] M. Brady, J. Hollerbach, T. Johnson, T. Lozano-Pérez and M. T. Masson. Robot motion: Planning and control. MIT Press, Cambridge (MA), 1983. (Cited in page 5.)

[Bretl 2004] T. Bretl, S. Rock, J.-C. Latombe, B. Kennedy and H. Aghazarian. *Free-Climbing with a Multi-Use Robot.* In Marcelo H. Ang Jr. and Oussama Khatib, editors, ISER, volume 21 of *Springer Tracts in Advanced Robot.*, pages 449–458. Springer, 2004. (Cited in pages 41 and 64.)

[Bretl 2008] T. Bretl and S. Lall. *Testing Static Equilibrium for Legged Robots.* IEEE Transactions on Robotics, vol. 24, no. 4, pages 794–807, 2008. (Cited in page 63.)

[Brock 2002] O. Brock and O. Khatib. *Elastic Strips: A Framework for Motion Generation in Human Environments.* International Journal of Robotics Research (IJRR), vol. 21, no. 12, pages 1031–1052, 2002. (Cited in page 8.)

[Caron 2015] S. Caron, Q.-C. Pham and Y. Nakamura. *Leveraging Cone Double Description for Multi-contact Stability of Humanoids with Applications to Statics and Dynamics.* In Robotics: Science and System, 2015. (Cited in page 97.)

[Carón 2016] S. Carón and A. Kheddar. *Multi-contact walking pattern generation based on model preview control of 3D COM accelerations.* In IEEE International Conference on Humanoid Robotics (Humanoids), pages 550–557, 2016. (Cited in page 63.)

[Chapuis 1949] A. Chapuis and E. Droz. Les Automates: figures artificielles d'hommes et d'animaux. Histoire et technique. Neuchatel, Editions du Griffon, 1949. (Cited in page 1.)

[Cheng 2008] K.B. Cheng, C.-H. Wang, H.-C. Chen, C.-D. Wu and H.-T. Chiu. *The mechanisms that enable arm motion to enhance vertical jump performance - A simulation study.* Journal of Biomechanics, vol. 41, no. 9, pages 1847–1854, 2008. (Cited in page 80.)

[Choi 2003] M.G. Choi, J. Lee and S.Y. Shin. *Planning biped locomotion using motion capture data and probabilistic roadmaps.* ACM Transactions on Graphics (TOG), vol. 22, no. 2, pages 182–203, 2003. (Cited in page 39.)

[Coros 2010] S. Coros, P. Beaudoin and M. van de Panne. *Generalized Biped Walking Control.* ACM Transactions on Graphics (Proceedings SIGGRAPH), vol. 29, no. 4, pages 130:1–130:9, 2010. (Cited in page 40.)

[Coros 2011] S. Coros, A. Karpathy, B. Jones, L. Reveret and M. van de Panne. *Locomotion Skills for Simulated Quadrupeds.* ACM Transactions on Graphics (TOG), vol. 30, no. 4, page Article TBD, 2011. (Cited in page 40.)

[Degani 2014] A. Degani, A.W. Long, S. Feng, H.B. Brown, R.D. Gregg, H. Choset, M.T. Mason and K.M. Lynch. *Design and Open-Loop Control of the ParkourBot, a Dynamic Climbing Robot.* IEEE Transactions on Robotics, vol. 30, no. 3, pages 705–718, 2014. (Cited in page 38.)

[Dellin 2012] C. Dellin and S. Srinivasa. *A framework for extreme locomotion planning.* In IEEE International Conference on Robotics and Automation (ICRA), pages 989–996, Saint Paul (MN), 2012. (Cited in page 40.)

[Edwardes 2009] D. Edwardes. The parkour and freerunning handbook, first edition. HarperCollins Publisher, 2009. (Cited in page 37.)

[Escande 2008] A. Escande, A. Kheddar, S. Miossec and S. Garsault. *Planning Support Contact-Points for Acyclic Motions and Experiments on HRP-2.* In Oussama Khatib, Vijay Kumar and George J Pappas, editors, ISER, volume 54 of *Springer Tracts in Advanced Robot.*, pages 293–302. Springer, 2008. (Cited in page 41.)

[Esteves 2006] C. Esteves, G. Arechavaleta, J. Pettré and J.-P. Laumond. *Animation planning for virtual mannequins cooperation.* ACM Transactions on Graphics (TOG), vol. 25, no. 2, pages 319–339, 2006. (Cited in page 39.)

[Fernbach 2017] P. Fernbach, S. Tonneau, A. Del Prete and M. Taïx. *A Kinodynamic steering-method for legged multi-contact locomotion.* In To appear in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2017. (Cited in page 80.)

[Garber 2004] M. Garber and M.C. Lin. Algorithmic Foundations of Robotics V, chapitre Constraint-Based Motion Planning Using Voronoi Diagrams, pages 541–558. Springer Berlin, 2004. (Cited in page 7.)

[Geraerts 2007] R. Geraerts and M. Overmars. *Creating High-Quality Paths for Motion Planning.* International Journal of Robotics Research (IJRR), vol. 26, no. 8, pages 845–863, 2007. (Cited in pages 8, 13, 14, 28 and 32.)

[Gilbert 1988] E.G. Gilbert, D.W. Johnson and S.S. Keerthi. *A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space.* IEEE J. Rob. Autom., vol. 4, no. 2, pages 193–203, 1988. (Cited in page 8.)

[Guernane 2011] R. Guernane and N. Achour. *Generating optimized paths for motion planning.* Robotics and Autonomous Systems, vol. 59, no. 10, pages 789–800, 2011. (Cited in page 8.)

[Ha 2012] S. Ha, Y. Ye and C.K. Liu. *Falling and Landing Motion Control for Character Animation.* ACM Transactions on Graphics (TOG), vol. 31, no. 6, page 1, 2012. (Cited in page 41.)

[Haldane 2016] D.W. Haldane, M.M. Plecnik, J.K. Yim and R.S. Fearing. *Robotic vertical jumping agility via series-elastic power modulation.* Science Robotics, vol. 1, no. 1, 2016. (Cited in page 38.)

[Hauser 2010] K. Hauser and V. Ng-Thow-Hing. *Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts.* In IEEE International Conference on Robotics and Automation (ICRA), pages 2493–2498, Anchorage (AK), 2010. (Cited in pages 8 and 13.)

[Hickox 2016] L.J. Hickox, B.M. Ashby and G.J. Alderink. *Exploration of the validity of the two-dimensional sagittal plane assumption in modeling the standing long jump.* Journal of Biomechanics, vol. 49, no. 7, pages 1085 – 1093, 2016. (Cited in page 65.)

[Hodgins 1995] J.K. Hodgins, W.L. Wooten, D.C. Brogan and J.F. O'Brien. *Animating Human Athletics.* In Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '95, pages 71–78, New York, NY, 1995. ACM. (Cited in page 40.)

[Holden 2016] D. Holden, J. Saito and T. Komura. *A Deep Learning Framework for Character Motion Synthesis and Editing.* ACM Transactions on Graphics (Proceedings SIGGRAPH), vol. 35, no. 4, pages 138:1–138:11, 2016. (Cited in page 39.)

[Kajita 2003] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi and H. Hirukawa. *Biped Walking Pattern Generation by using Preview Control of Zero-Moment Point.* In IEEE International Conference on Robotics and Automation (ICRA), Taipei, Taiwan, 2003. (Cited in page 39.)

[Kalakrishnan 2011] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor and S. Schaal. *STOMP: Stochastic trajectory optimization for motion planning.* In IEEE International Conference on Robotics and Automation (ICRA), pages 4569–4574, Shanghai, China, 2011. (Cited in page 7.)

[Kane 1969] T.R. Kane and M.P. Scher. *A dynamical explanation of the falling cat phenomenon.* International Journal of Solids and Structures, vol. 5, no. 7, pages 663–670, 1969. (Cited in pages 80 and 83.)

[Kapadia 2016] M. Kapadia, X. Xianghao, M. Nitti, M. Kallmann, S. Coros, R. W. Sumner and M. Gross. *Precision: Precomputing Environment Semantics for Contact-rich Character Animation.* In Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, I3D '16, pages 29–37, New York, NY, USA, 2016. (Cited in page 39.)

[Karaman 2011] S. Karaman and E. Frazzoli. *Sampling-based Algorithms for Optimal Motion Planning.* International Journal of Robotics Research (IJRR), vol. 30, no. 7, pages 846–894, 2011. (Cited in page 7.)

[Kavraki 1996] L.E. Kavraki, P. Svestka, J.-C. Latombe and M.H. Overmars. *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*. IEEE Trans. Rob. Autom., vol. 12, no. 4, pages 566–580, 1996. (Cited in pages 6, 53 and 61.)

[Kovar 2002] L. Kovar, M. Gleicher and F. Pighin. *Motion graphs*. In ACM Transactions on Graphics (TOG), volume 21, New York, NY, USA, 2002. ACM. (Cited in page 39.)

[Kry 2009] P. Kry, L. Reveret, F. Faure and M.-P. Cani. *Modal locomotion: animating virtual characters with natural vibrations*. Computer Graphics Forum, vol. 28, no. 2, pages 289–298, 2009. (Cited in page 41.)

[Kuffner 2000] J.J. Kuffner and S.M. Lavalle. *RRT-Connect: An efficient approach to single-query path planning*. In IEEE International Conference on Robotics and Automation (ICRA), volume 2, pages 995–1001, San Francisco (CA), 2000. (Cited in page 25.)

[Kunz 2014] T. Kunz and M. Stilman. *Probabilistically complete kinodynamic planning for robot manipulators with acceleration limits*. In IEEE International Conference on Intelligent Robots and Systems (IROS), pages 3713–3719, 2014. (Cited in pages 54 and 62.)

[Lamouret 1996] A. Lamouret and M. van de Panne. Motion synthesis by example, pages 199–212. Springer Vienna, Vienna, 1996. (Cited in page 40.)

[Latombe 1991] J.-C. Latombe. Robot motion planning. Kluwer Academic Publishers, Boston, MA, 1991. (Cited in page 5.)

[Laumond 2006] J.-P. Laumond. *A success story of motion planning algorithms*. IEEE Robotics and Automation Magazine, vol. 13, no. 2, 2006. (Cited in page 9.)

[Laumond 2014] J.-P. Laumond, N. Mansard and J.-B. Lasserre. *Optimality in Robot Motion: Optimal Versus Optimized Motion*. Communications of the ACM, vol. 57, no. 9, pages 82–89, 2014. (Cited in page 15.)

[LaValle 2001] S.M. LaValle and J.J. Kuffner, Jr. Algorithmic and computational robotics: New directions, chapitre Rapidly-Exploring Random Trees: Progress and Prospects, pages 293–308. Wellesley (MA), 2001. (Cited in page 6.)

[LaValle 2006] S.M. LaValle. Planning algorithms. Cambridge University Press, New York, NY, USA, 2006. (Cited in page 5.)

[Levine 2011] S. Levine, Y. Lee, V. Koltun and Z. Popović. *Space-time Planning with Parameterized Locomotion Controllers*. ACM Transactions on Graphics (TOG), vol. 30, no. 3, 2011. (Cited in page 40.)

[Levine 2012] S. Levine and J. Popović. *Physically Plausible Simulation for Character Animation*. In Symposium on Computer Animation, pages 221–230, 2012. (Cited in page 41.)

[Liu 2012] L. Liu, K. Yin, M. van de Panne and B. Guo. *Terrain Runner: Control, Parameterization, Composition, and Planning for Highly Dynamic Motions*. ACM Transactions on Graphics (Proceedings SIGGRAPH), vol. 31, no. 6, pages 154:1–154:10, 2012. (Cited in page 40.)

[Liu 2016] L. Liu, M. van de Panne and K. Yin. *Guided learning of control graphs for physics-based characters*. ACM Transactions on Graphics (TOG), vol. 35, no. 3, page 29, 2016. (Cited in page 40.)

[Lozano-Pérez 1983] T. Lozano-Pérez. *Spatial Planning: A Configuration Space Approach*. IEEE Trans. on Computers, vol. C-32, pages 108–120, 1983. (Cited in page 5.)

[Mather 2009] T.W. Mather and M. Yim. *Modular configuration design for a controlled fall*. In 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 5905–5910, 2009. (Cited in pages 80 and 83.)

[McCann 2006] J. McCann, N.S. Pollard and S. Srinivasa. *Physics-Based Motion Retiming*. In ACM SIGGRAPH / Eurographics Symposium on Computer Animation, 2006. (Cited in page 40.)

[Mirabel 2016] J. Mirabel, S. Tonneau, P. Fernbach, A. Seppälä, M. Campana, N. Mansard and F. Lamiraux. *HPP: a new software for constrained motion planning*. In IEEE International Conference on Intelligent Robots and Systems (IROS), 2016. (Cited in page 9.)

[Montgomery 1993] R. Montgomery. Gauge theory of the falling cat, volume 1. Fields Institute Communications, 1993. (Cited in pages 80 and 83.)

[Mordatch 2012] I. Mordatch, E. Todorov and Z. Popović. *Discovery of complex behaviors through contact-invariant optimization*. ACM Transactions on Graphics (TOG), vol. 31, no. 4, 2012. (Cited in pages 40 and 41.)

[Nocedal 2006] J. Nocedal and S. Wright. Numerical optimization. Springer New York, 2nd ed., 2006. (Cited in page 20.)

[Orin 2013] D. Orin, A. Goswami and S.H. Lee. *Centroidal dynamics of a humanoid robot*. Autonomous Robots, vol. 35, no. 2, pages 161–176, 2013. (Cited in page 97.)

[Pan 2012a] J. Pan, S. Chitta and D. Manocha. *FCL: A general purpose library for collision and proximity queries*. In IEEE International Conference on Robotics and Automation (ICRA), pages 3859–3866, Saint Paul (MN), 2012. (Cited in page 19.)

[Pan 2012b] J. Pan, L. Zhang and D. Manocha. *Collision-free and Smooth Trajectory Computation in Cluttered Environments*. International Journal of Robotics Research (IJRR), vol. 31, no. 10, pages 1155–1175, 2012. (Cited in page 8.)

[Papadopoulos 2007] E. Papadopoulos, I. Fragkos and I. Tortopidis. *On Robot Gymnastics Planning with Non-zero Angular Momentum*. In IEEE International Conference on Robotics and Automation (ICRA), pages 1443–1448, 2007. (Cited in page 38.)

[Park 2012] C. Park, J. Pan and D. Manocha. *ITOMP: Incremental Trajectory Optimization for Real-Time Replanning in Dynamic Environments*. In International Conference on Automated Planning and Scheduling (ICAPS), pages 207–215, Atibaia, São Paulo, Brazil, 2012. (Cited in page 7.)

[Peng 2016] X.B. Peng, G. Berseth and M. van de Panne. *Terrain-Adaptive Locomotion Skills Using Deep Reinforcement Learning*. ACM Transactions on Graphics (Proceedings SIGGRAPH), vol. 35, no. 5, 2016. (Cited in page 40.)

[Pettré 2003] J. Pettré, J.-P. Laumond and T. Siméon. *A 2-stages Locomotion Planner for Digital Actors*. In Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '03, pages 258–264, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association. (Cited in page 39.)

[Pollard 2000] N.S. Pollard and F. Behmaram-Mosavat. *Force-based motion editing for locomotion tasks*. In IEEE International Conference on Robotics and Automation (ICRA), volume 1, pages 663–669 vol.1, 2000. (Cited in page 40.)

[Raibert 1984] M.H. Raibert, M.A. Chepponis and H. Benjamin Brown. *Experiments in Balance With a 3D One-Legged Hopping Machine*. International Journal of Robotics Research (IJRR), vol. 3, pages 75–92, 1984. (Cited in page 38.)

[Reitsma 2003] P.S.A. Reitsma and N.S. Pollard. *Perceptual Metrics for Character Animation: Sensitivity to Errors in Ballistic Motion*. ACM Transactions on Graphics (TOG), vol. 22, no. 3, pages 537–542, 2003. (Cited in page 41.)

[Reitsma 2008] P.S.A. Reitsma, J. Andrews and N.S. Pollard. *Effect of Character Animacy and Preparatory Motion on Perceptual Magnitude of Errors in Ballistic Motion*. Comput. Graph. Forum, vol. 27, no. 2, pages 201–210, 2008. (Cited in page 41.)

[Rose 1996] C. Rose, B. Guenter, B. Bodenheimer and M.F. Cohen. *Efficient Generation of Motion Transitions Using Spacetime Constraints*. In Proceedings

of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '96, pages 147–154, New York, NY, USA, 1996. ACM. (Cited in page 40.)

[Safonova 2004] A. Safonova, J.K. Hodgins and N.S. Pollard. *Synthesizing Physically Realistic Human Motion in Low-dimensional, Behavior-specific Spaces.* ACM Transactions on Graphics (TOG), vol. 23, no. 3, pages 514–521, 2004. (Cited in page 40.)

[Safonova 2007] A. Safonova and J.K. Hodgins. *Construction and Optimal Search of Interpolated Motion Graphs.* ACM Transactions on Graphics (TOG), vol. 26, no. 3, 2007. (Cited in page 40.)

[Schulman 2014] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg and P. Abbeel. *Motion Planning with Sequential Convex Optimization and Convex Collision Checking.* International Journal of Robotics Research (IJRR), vol. 33, no. 9, pages 1251–1270, 2014. (Cited in pages 8 and 15.)

[Schwartz 1983] J.T. Schwartz and M. Sharir. *On the piano movers problem I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers.* Communications on Pure and Applied Mathematics, vol. 36, no. 3, pages 345–398, 1983. (Cited in page 5.)

[Schwarzer 2004] F. Schwarzer, M. Saha and J.-C. Latombe. Algorithmic Foundations of Robotics V, chapitre Exact Collision Checking of Robot Paths, pages 25–41. Springer Berlin, 2004. (Cited in page 16.)

[Sekhavat 1998] S. Sekhavat, P. Svestka, J.-P. Laumond and M. Overmars. *Multilevel path planning for nonholonomic robots using semi-holonomic subsystems.* International Journal of Robotics Research (IJRR), vol. 17, no. 8, pages 840–857, 1998. (Cited in pages 13 and 27.)

[Shapiro 2011] A. Shapiro and S.H. Lee. *Practical Character Physics For Animators.* Computer Graphics and Applications, Special Issue on Physics, vol. 31, no. 4, pages 45–55, 2011. (Cited in pages 40 and 41.)

[Shu 2016] R. Shu, A. Siravuru, A. Rai, T. Dear, K. Sreenath and H. Choset. *Optimal Control for Geometric Motion Planning of a Robot Diver.* In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2016. (Cited in pages 80 and 83.)

[Siméon 2000] T. Siméon, J.-P. Laumond and C. Nissoux. *Visibility based probabilistic roadmaps for motion planning.* Advanced Robotics Journal, vol. 14, no. 6, pages 477–493, 2000. (Cited in pages 25 and 57.)

[Stoeter 2005] S.A. Stoeter and N. Papanikolopoulos. *Autonomous stair-climbing with miniature jumping robots.* IEEE Transactions on Systems, Man, and Cybernetics: Part B, vol. 35, no. 2, pages 313–325, 2005. (Cited in page 38.)

[Sulejmanpašić 2005] A. Sulejmanpašić and J. Popović. *Adaptation of Performed Ballistic Motion.* ACM Transactions on Graphics (TOG), vol. 24, no. 1, pages 165–179, 2005. (Cited in page 41.)

[Tonneau 2015a] S. Tonneau. *Motion planning and synthesis for virtual characters in constrained environments.* PhD thesis, Institut National des Sciences Appliquées (INSA) Rennes, France, 2015. (Cited in page 67.)

[Tonneau 2015b] S. Tonneau, N. Mansard, C. Park, D. Manocha, F. Multon and J. Pettré. *A reachability-based planner for sequences of acyclic contacts in cluttered environments.* In International Symposium on Robotics Research (ISRR), 2015. (Cited in pages 62, 65, 67 and 71.)

[Tonneau 2016a] S. Tonneau, R. A. Al-Ashqar, J. Pettré, T. Komura and N. Mansard. *Contact re-positioning under large environment deformation.* Computer Graphics Forum (Proc. Eurographics), 2016. (Cited in page 41.)

[Tonneau 2016b] S. Tonneau, A. Del Prete, J. Pettré, C. Park, D. Manocha and N. Mansard. *An efficient acyclic contact planner for multiped robots.* Technical report, 2016. (Cited in page 79.)

[van Basten 2011] B.J.H. van Basten, A. Egges and R. Geraerts. *Combining path planners and motion graphs.* Journal Visualization and Computer Animation, vol. 22, no. 1, pages 59–78, 2011. (Cited in page 39.)

[Wensing 2014] P.M. Wensing and D.E. Orin. *Development of high-span running long jumps for humanoids.* In IEEE International Conference on Robotics and Automation (ICRA), pages 222–227, 2014. (Cited in page 41.)

[Witkin 1988] A. Witkin and M. Kass. *Spacetime constraints.* ACM SIGGRAPH Computer Graphics, vol. 22, no. 4, pages 159–168, 1988. (Cited in page 40.)

[Witkin 1995] A. Witkin and Z. Popović. *Motion Warping.* In Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '95, pages 105–108, New York, NY, 1995. ACM. (Cited in page 39.)

[Wooten 1996] W.L. Wooten and J.K. Hodgins. *Animation of Human Diving.* Computer Graphics Forum, 1996. (Cited in page 40.)

[Yamane 2010] K. Yamane and K. W. Sok. *Planning and Synthesizing Superhero Motions.* In Conference on Motion in Games, pages 254–265, 2010. (Cited in pages 39 and 41.)

[Yin 2007] K. Yin, K. Loken and M. van de Panne. *SIMBICON: Simple Biped Locomotion Control*. ACM Transactions on Graphics (TOG), vol. 26, no. 3, 2007. (Cited in page 40.)

[Zhao 2015] J. Zhao, T. Zhao, N. Xi, M. W. Mutka and L. Xiao. *MSU Tailbot: Controlling Aerial Maneuver of a Miniature-Tailed Jumping Robot*. IEEE/ASME Transactions on Mechatronics, vol. 20, no. 6, pages 2903–2914, 2015. (Cited in pages 80 and 83.)

[Zucker 2013] M. Zucker, N. Ratliff, A. Dragan, M. Pivtoraiko, M. Klingensmith, C. Dellin, J. Bagnell and S. Srinivasa. *CHOMP: Covariant Hamiltonian optimization for motion planning*. International Journal of Robotics Research (IJRR), vol. 32, no. 9-10, pages 1164–1193, 2013. (Cited in pages 7, 14 and 15.)

Résumé en Français :

Les algorithmes probabilistes offrent de puissantes possibilités quant à la résolution de problèmes de planification de mouvements pour des robots complexes dans des environnements quelconques. Cependant, la qualité des chemins solutions obtenus est discutable. Cette thèse propose un outil pour optimiser ces chemins et en améliorer la qualité. La méthode se base sur l'optimisation numérique contrainte et la détection de collision pour réduire la longueur du chemin tout en évitant les collisions.

La modularité des méthodes probabilistes nous a aussi inspirés pour réaliser un algorithme de génération de sauts pour des personnages. Cet algorithme est décrit par trois étapes de planifications, de la trajectoire du centre du personnage jusqu'à son mouvement corps-complet. Chaque étape bénéficie de la rigueur de la planification pour éviter les collisions et pour contraindre le chemin. Nous avons proposé des contraintes inspirées de la physique pour améliorer la plausibilité des mouvements, telles que du non-glissement, de la limitation de vitesse et du maintien de contacts.

Les travaux de cette thèse ont été intégrés dans le logiciel "Humanoid Path Planner" et les rendus visuels effectués avec Blender.

Mots clés : Planification de mouvement, animation graphique, mouvement ballistique, optimisation de chemin, simulation

Abstract:

Probabilistic algorithms offer powerful possibilities as for solving motion planning problems for complex robots in arbitrary environments. However, the quality of obtained solution paths is questionable. This thesis presents a tool to optimize these paths and improve their quality. The method is based on constrained numerical optimization and on collision checking to reduce the path length while avoiding collisions.

The modularity of probabilistic methods also inspired us to design a motion generation algorithm for jumping characters. This algorithm is described by three steps of motion planning, from the trajectory of the character's center to the wholebody motion. Each step benefits from the rigor of motion planning to avoid collisions and to constraint the path. We proposed physics-inspired constraints to increase the plausibility of motions, such as slipping avoidance, velocity limitation and contact maintaining.

The thesis works have been implemented in the software 'Humanoid Path Planner' and the graphical renderings have been done with Blender.

Keywords: Motion planning, computer animation, ballistic motion, path optimization, simulation